Evolution von Ontologien in den Lebenswissenschaften

Von der Fakultät für Mathematik und Informatik
der Universität Leipzig
angenommene

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

DOCTOR RERUM NATURALIUM (Dr. rer. nat.)

im Fachgebiet Informatik

vorgelegt von

Diplom-Informatiker Michael Hartung

geboren am 2. Oktober 1980 in Sonneberg

Die Annahme der Dissertation haben empfohlen:

1. Prof. Dr. Erhard Rahm (Universität Leipzig)
2. Prof. Dr. York Sure (Universität Koblenz–Landau)

Die Verleihung des akademischen Grades erfolgt mit Bestehen der Verteidigung am 20. April 2011 mit dem Gesamtprädikat *magna cum laude*.

Danksagung

Die vorliegende Dissertation entstand innerhalb der letzten 5 Jahre während meiner Tätigkeit am Interdisziplinären Zentrum für Bioinformatik (IZBI) sowie in der Abteilung Datenbanken am Institut für Informatik der Universität Leipzig. Mein größter Dank gilt meinem Betreuer und Mentor Herrn Prof. Dr. Erhard Rahm. Er gab mir die Möglichkeit zur Promotion und stand mir in den vergangenen Jahren stets mit Rat und Tat zur Seite. Hierzu zählen insbesondere die unzähligen Gespräche und Diskussionen über meine Ideen und Ergebnisse sowie die hilfreichen Verbesserungsvorschläge bei der Erstellung von wissenschaftlichen Publikationen.

Diese Arbeit wäre ohne die finanzielle Unterstützung des Bundesministeriums für Bildung und Forschung (BMBF) sowie der Deutschen Forschungsgemeinschaft (DFG) nicht möglich gewesen. Diesbezüglich möchte ich mich bei Herrn Prof. Dr. Markus Löffler (wissenschaftlicher Leiter des IZBI) sowie bei Herrn PD Dr. Hans Binder (Geschäftsführer des IZBI) für die reibungslose Finanzierung meiner Mitarbeiterstelle im MediGRID Projekt bedanken. Zudem konnte ich durch die Finanzierung von Konferenzreisen und Publikationskosten meine wissenschaftlichen Ergebnisse international präsentieren und publizieren.

Des Weiteren möchte ich mich bei allen Kollegen der Abteilung Datenbanken am Institut für Informatik für das hervorragende Arbeitsklima bedanken. Insbesondere die Erlebnisse während unserer jährlichen Seminare in Zingst werden mir immer in Erinnerung bleiben. Ein ganz besonderer Dank gilt Frau Anika Groß und Herrn Dr. Toralf Kirsten für die tolle und kreative Zusammenarbeit innerhalb unserer gemeinsamen Projekte. Ohne die beiden wären einige in dieser Arbeit beschriebenen Ergebnisse nicht möglich gewesen. Herrn Dr. Toralf Kirsten, mit dem ich in den ersten Jahren ein Zimmer am IZBI teilte, danke ich besonders für die Unterstützung während meiner Anfangsphase in Leipzig sowie für seine Ideen zur Konzeption und Realisierung der Ontologieversionierung. Bei Frau Anika Groß, welche 2008 unsere Arbeitsgruppe ergänzte und mit der ich $2^1/_2$ Jahre ein Zimmer in der Johannisgasse teilen durfte, möchte ich mich insbesondere für die unzähligen Verbesserungsvorschläge und Evaluierungen für meine Arbeiten bedanken. Unvergesslich werden mir die gemeinsamen Ausarbeitungen an unserem Whiteboard sowie das kollaborative Editieren unserer Publikationen inklusive Chat, Telefon usw. bleiben. Ein ebenso herzlichster Dank gilt den Frauen Corinna Dittmann, Andrea Hesse und Petra Pre-

gel sowie Herrn Jens Steuck, welche mir bei der Durchführung von administrativen Aufgaben stets hilfreich zur Seite standen.

Für das Korrekturlesen und die kritische Durchsicht dieser Arbeit bin ich Anika Groß zutiefst dankbar. Ihre Hinweise und Vorschläge halfen mir Fehler aufzudecken und ich konnte somit die Arbeit sukzessive verbessern. Ein herzlicher Dank gilt ebenfalls den Gutachtern der Dissertation, welche durch ihre Hinweise zu einer Verbesserung der Arbeit beigetragen haben.

Ein besonderer Dank gilt natürlich meinen Eltern Marion und Günter Hartung sowie meiner Schwester Carolin, dir mir auf meinem bisherigen Weg stets unterstützend zur Seite standen.

Leipzig, den 11. Januar 2011 Michael Hartung

Zusammenfassung

In den *Lebenswissenschaften* haben sich *Ontologien* in den letzten Jahren auf breiter Front durchgesetzt und sind in vielen Anwendungs- und Analyseszenarien kaum mehr wegzudenken. So etablierten sich nach und nach immer mehr domänenspezifische Ontologien, z. B. Anatomie-Ontologien oder Ontologien zur Beschreibung der Funktionen von Genen oder Proteinen. Da das Wissen in den Lebenswissenschaften sich rapide ändert und weiterentwickelt, müssen die entsprechenden Ontologien ständig angepasst und verändert werden, um einen möglichst aktuellen Wissensstand zu repräsentieren. Nutzer von Ontologien müssen mit dieser *Evolution* umgehen können, d. h. um „Up-to-Date“ zu sein, sollten die aktuellsten Versionen einer Ontologie verwendet werden. Dies ist häufig nur schwer umsetzbar, da die Evolution weitreichende Einflüsse auf existierende Datenbestände, Analyseergebnisse oder Anwendungen haben kann. Innerhalb dieser Dissertation stehen Werkzeuge und Algorithmen zum Umgang mit sich ständig ändernden Ontologien im Bereich der Lebenswissenschaften im Mittelpunkt.

Zunächst wird ein generelles Framework für *quantitative Evolutionsanalysen* eingeführt. Das Framework wird für eine umfassende Analyse der Evolution zahlreicher Ontologien der Lebenswissenschaften verwendet. Die Analysen zeigen, dass alle untersuchten Ontologien stetig verändert (angepasst) werden und ein signifikantes Wachstum aufweisen. Auch für auf Ontologien basierte Mappings, d. h. Verknüpfungen zwischen Datenquellen und Ontologien (Annotation-Mapping) sowie zwischen Ontologien selbst (Ontologie-Mapping), liegen starke und häufige Veränderungen vor. Es besteht somit ein Bedarf, die Evolution von Ontologien in den Lebenswissenschaften und deren Konsequenzen zu unterstützen, d. h. Nutzern von sich ständig ändernden Ontologien angemessene Algorithmen/Werkzeuge bereitzustellen. Die Erkenntnisse aus den durchgeführten Analysen bilden die Basis für die nachfolgenden Arbeiten.

Eine immer wiederkehrende Aufgabe im Rahmen der Ontologieevolution besteht in der Bestimmung von Änderungen zwischen zwei Versionen einer Ontologie, d. h. worin besteht der Unterschied und wie hat sich die neuere Version aus der alten Version heraus entwickelt. Das Ergebnis, d. h. der *Diff* (die Differenz) zwischen den beiden Ontologieversionen, bildet die Basis für weitere Aufgaben wie beispielsweise die Anpassung abhängiger Daten. Innerhalb der Arbeit wird ein neuartiger auf Regeln

basierter Algorithmus vorgestellt, welcher den Diff zwischen zwei Ontologieversionen bestimmt. Es werden sowohl einfache wie auch komplexe Änderungen erkannt, was eine kompakte, intuitive und verständliche Diff-Repräsentation garantiert. Es wird theoretisch wie praktisch gezeigt, dass ein vollständiger Diff bestimmt wird, was eine korrekte Migration von Ontologieversionen ermöglicht.

Ein weiterer Schwerpunkt der Arbeit liegt in der Bestimmung änderungsintensiver bzw. stabiler Regionen in einer Ontologie. Dazu wird die Notation von *Ontologieregionen* und zugehörige Metriken zur Beurteilung ihrer Änderungsintensität (Stabilität) eingeführt. Ein neuartiger automatisierter Algorithmus erlaubt die Bestimmung (in)stabiler Ontologieregionen auf Basis veröffentlichter Ontologieversionen in einem vorgegebenen Zeitraum. Durch erkannte Änderungen zwischen Ontologieversionen und mit Hilfe der Ontologiestruktur werden änderungsintensive bzw. stabile Ontologieregionen erkannt. Die Evaluierung anhand großer Ontologien der Lebenswissenschaften zeigt, dass der Algorithmus in der Lage ist (in)stabile Ontologieregionen automatisiert zu bestimmen.

Abschließend wird das webbasierte System *OnEX* und dessen *Versionierungsansatz* präsentiert. OnEX ermöglicht einen benutzerfreundlichen und interaktiven Zugang zu Informationen über die Evolution und Änderungen in Ontologien der Lebenswissenschaften. Nutzer können Ontologien aus ihrem Interessengebiet bzgl. Evolution untersuchen, indem sie beispielsweise Änderungen an einer Ontologieversion einsehen, welche in einer Analyse oder Anwendung genutzt werden soll. Der OnEX zugrunde liegende Versionierungsansatz ermöglicht eine skalierbare und speichereffiziente Versionierung großer Ontologien durch die Nutzung von Zeitstempeln. Mit Hilfe des Ansatzes konnten 16 Ontologien mit ca. 700 Versionen seit 2002 versioniert und Nutzern über OnEX für Evolutionsanalysen zugänglich gemacht werden.

Inhaltsverzeichnis

Teil I

Einleitung

1 Einleitung

1.1 Einführung

Ontologien haben durch ihre Verwendung in verschiedenen Wissenschaftsdisziplinen in den letzten Jahren enorm an Bedeutung gewonnen. Sie ermöglichen die Konzeptualisierung einer Domäne [39] und bestehen aus einer Menge sogenannter Konzepte (Kategorien), welche ein fest vereinbartes, gemeinsames Vokabular bilden. Ihre Hauptanwendung besteht in der einheitlichen und konsistenten Beschreibung (Annotation) von Objekten (Instanzen) einer Domäne. Auf Basis der Ausdrucksstärke einer Ontologie können einfache Vokabulare mit verbalen Beschreibungen der Konzepte, Taxonomien und Thesauri sowie komplexe Ontologien mit einer formalen axiomatischen Definition unterschieden werden [69]. Abb. 1.1 zeigt das Spektrum existierender Ontologien mit ansteigender Ausdrucksfähigkeit. Am linken Ende der Skala befinden sich einfache Kataloge. Diese bestehen typischerweise aus einer Menge unstrukturierter Terme einer Domäne. Die Terme besitzen weder Definitionen noch sind detaillierte Beschreibungen verfügbar. In Glossaren werden den Termen Bedeutungen in natürlicher Sprache zugewiesen. Die etwas ausdrucksstärkeren Thesauri fügen zusätzlich semantisches Wissen in Form von Synonymen ein, eine explizite Hierarchie (Struktur) ist nicht vorhanden. Die nachfolgenden is_a Hierarchien führen Spezialisierungen für Terme ein, d. h. die Terme werden strukturiert angeordnet. Es wird zwischen informalen und formalen is_a Hierarchien unterschieden. Bei informalen is_a Hierarchien wird keine Garantie gegeben, dass eine Instanz eines spezifischen Konzepts ebenfalls Instanz eines entsprechend der Hierarchie generelleren Konzepts sein muss. In formalen Hierarchien hingegen werden die is_a

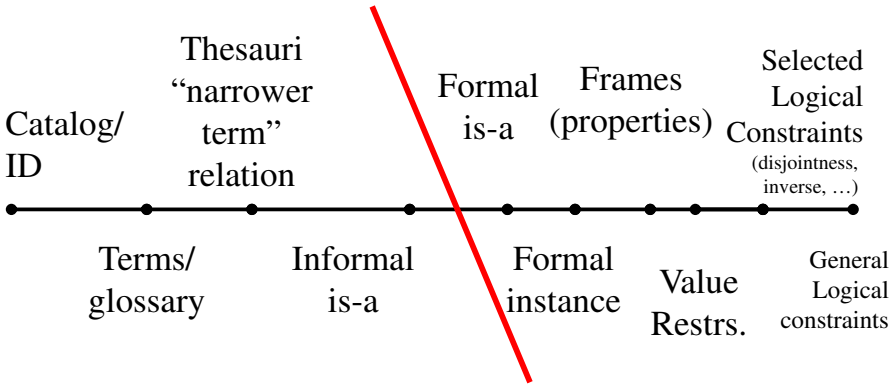


Abbildung 1.1: Ontologiespektrum nach [69]

Beziehungen zwischen den Konzepten als strikt angesehen, d. h. eine Instanz eines Konzepts ist ebenfalls Instanz aller Vorgängerkonzepte. Im weiteren Verlauf des Spektrums werden sukzessive zusätzliche Konstrukte wie Frames (Properties), spezifische Beziehungen wie part_of, Disjunktheitsbeziehungen oder Restriktionen zur Modellierung von Domänenwissen herangezogen. In dieser Arbeit stehen die in wissenschaftlichen Anwendungen häufig genutzten, strukturierten Ontologien im Mittelpunkt. Die Konzepte beschrieben über Attribute wie Name oder Synonyme sind dabei mit semantisch gerichteten Beziehungen (z. B. is_a oder part_of) untereinander verbunden und bilden eine baum- bzw. graphartige Struktur aus.

Insbesondere in den Lebenswissenschaften haben sich Ontologien etabliert [4, 13, 68] und sind heutzutage in Anwendungen, zur Beschreibung von Experimenten oder in Analysen kaum mehr wegzudenken. Die enorme Wichtigkeit zeigt sich u. a. in der hohen Anzahl verfügbarer Ontologien. So werden beispielsweise in der OBO-Foundry[1] [110] oder im BioPortal[2] [91] über 200 verschiedene Ontologien verwaltet und Anwendern bereitgestellt. Die wohl am weitesten verbreitete und genutzte Ontologie ist die Gene Ontology (GO) [3, 36], welche über ihre drei Subontologien Prozesse, Funktionen und Komponenten eine konsistente und semantisch einheitliche Beschreibung von molekular-biologischen Objekten ermöglicht. So nutzen viele verfügbare Biodatenbanken wie Ensembl [33], SwissProt [14] oder RefSeq [103] GO zur Annotation der erfassten Proteine oder Gene, um terminologische Variationen und daraus folgende Fehlinterpretationen zu reduzieren [6]. Auch biomedizinische Literatur in PudMed[3] wird mittels MeSH (Medical Subject Headings) [72] einheitlich klassifiziert, was Nutzern erweiterte Such- und Navigationsmöglichkeiten eröffnet. Weitere Teilgebiete der Lebenswissenschaften, welche innerhalb von Ontologien modelliert werden sind u. a. Anatomien verschiedener Spezies [40, 54, 106, 111], Krankheiten [77, 95, 109] oder biochemische Strukturen [26]. Zudem werden Ontologien in

[1] The Open Biological and Biomedical Ontologies Foundry: `http://www.obofoundry.org`
[2] NCBO BioPortal: `http://bioportal.bioontology.org`
[3] PubMed: `http://pubmed.org`

internationalen Großprojekten wie dem caBIG [18] oder nationalen Forschungsverbünden wie dem MediGRID [52, 67] bzw. D-Grid [37, 50, 51] zur Standardisierung und Etablierung eines gemeinsamen Vokabulars verwendet.

Neben der wachsenden Anzahl von Ontologien und Ontologie-relevanten Arbeiten in den Lebenswissenschaften [13], besteht die Notwendigkeit existierende Ontologien zu ändern bzw. anzupassen, um somit einen möglichst aktuellen und korrekten Wissenstand zu repräsentieren (Evolution). Anpassungen an den Ontologien haben zahlreiche Gründe, z. B.:

- Integration von neuem/geändertem Domänenwissen, z. B. aus experimentellen Ergebnissen oder Analysen
- Behebung initialer Designfehler entstanden in früheren Versionen einer Ontologie
- Veränderte Anforderungen seitens der Nutzer oder Applikationen/Analysen, z. B. geänderte Nutzungsmuster oder der Bedarf neuartige Analysen zu unterstützen
- Umsetzung von Designrichtlinien, z. B. Umstrukturierung/Reorganisation des erfassten Wissens
- Migration zu einer anderen Ontologiesprache

Insbesondere der erste Punkt hängt stark mit der heutzutage weltweit vernetzten Forschung zusammen, d. h. es werden neue Erkenntnisse binnen kurzer Zeit produziert und publiziert. So werden viele Ontologien in den Lebenswissenschaften modifiziert, wenn beispielsweise neue experimentelle Ergebnisse oder Analyseresultate integriert werden müssen. Dies kann zur Einfügung von neuem Wissen wie z. B. neuer Konzepte führen. Jedoch kann bereits erfasstes Wissen auch revidiert werden, wenn beispielsweise neue Erkenntnisse die Verwerfung oder Modifikation eines Sachverhalts erfordern. Nach der Veränderung einer Ontologie aufgrund der genannten Gründe wird oftmals eine neue Ontologieversion veröffentlicht, welche dann von Anwendern genutzt werden kann.

Ontologieevolution umfasst zwar in erster Linie die Entwicklung von Ontologien selbst, allerdings haben die durchgeführten Änderungen auch weitreichende Auswirkungen auf Ontologie-basierte Datenquellen, Applikationen oder Analysen. So besteht bei der Veröffentlichung einer neuen Ontologieversion oftmals die Frage, ob eine Anwendung noch lauffähig ist oder ob Daten wie Annotationen, welche die Ontologie verwenden weiterhin Gültigkeit besitzen. Häufig werden Nutzer und Anwender mit diesen Problemen allein gelassen, d. h. es wird zwar eine neue, verbesserte Ontologieversion veröffentlicht, die resultierenden Anpassungen und Migrationen bleiben jedoch oftmals dem Endnutzer überlassen. Das Problem verstärkt sich,

falls wie in den Lebenswissenschaften hohe Veröffentlichungsfrequenzen zur Wahrung der Aktualität anzutreffen sind. Beispielsweise veröffentlicht das Konsortium der GO täglich eine neue Version der Gene Ontology. Andere Ontologien wie der NCI Thesaurus [109] geben jeden Monat eine neue Version frei.

Auch Ontologie-basierte Daten wie Mappings, welche Konzepte einer Ontologie in Korrespondenzen nutzen, unterliegen ständigen Änderungen. So können die Korrespondenzen (Annotationen) eines *Annotation-Mapping* zwischen Objekten einer Datenquelle und Konzepten einer Ontologie angepasst werden. Beispielsweise werden die Annotationen eines Proteins zu Konzepten der GO verändert, wenn in neuen Forschungsergebnissen Erkenntnisse über die molekularen Funktionen oder biologischen Prozesse des Proteins vorliegen. In ähnlicher Art und Weise können Produkte, welche in einem Produktkatalog klassifiziert sind (z. B. bei EBay[4] oder Amazon[5]) die Produktkategorie wechseln, falls beispielsweise ein Produkt besser vermarktet werden soll oder Umstrukturierungen des Katalogs eine Neueinordnung erfordern. Auch Korrespondenzen zwischen den Konzepten zweier Ontologien, sogenannte *Ontologie-Mappings*, benötigen Anpassungen, falls die beteiligten Ontologien sich verändern. So werden insbesondere zwischen Anatomieontologien verschiedener Spezies Ontologie-Mappings erstellt, um u. a. vergleichende Analysen oder die Übertragung von Wissen zu unterstützen. Ein Beispiel ist das Ontologie-Mapping zwischen der AdultMouseAnatomy [54] und dem Anatomieteil des NCI Thesaurus [109]. Mit Hilfe dieses Mappings können beispielsweise Erkenntnisse aus Experimenten an Mäusen entsprechend auf die Anatomie des Menschen übertragen werden. Eine Evolution, d. h. Veränderungen in den beiden Ontologien erfordern gegebenenfalls auch eine Anpassung des Mappings, da neu hinzugekommenes Wissen oder veränderte Konzepte dessen Gültigkeit wie auch Aktualität beeinflussen. Fallen derartige Änderungen häufig an, entstehen Instabilitäten in den Mappings. Diese können wiederum Folgen für Anwendungen oder Analysen haben, welche auf diesen Mappings basieren. So werden beispielsweise Annotationen in der Bestimmung von Proteinfunktionen (Functional Profiling) verwendet [16, 102], was bei auftretenden Instabilitäten zu Veränderungen in den Ergebnissen führen kann.

Die steigende Komplexität der Ontologien insbesondere in den Lebenswissenschaften lässt einen manuellen Umgang mit Evolution in der Regel nicht zu. Viele Ontologien umfassen mehrere zehntausend Konzepte (z. B. GO ≈30.000 oder NCI Thesaurus ≈70.000), welche miteinander vernetzt sind und somit komplexe Graphen ausbilden. Häufig existieren keine Informationen darüber, welche Teile oder Konzepte einer Ontologie bei einem Versionswechsel betroffen sind. Die Protokolle der durchgeführten Änderungen liegen meist nur den Entwicklern selbst vor und werden nicht veröffentlicht. Diese Tatsache erschwert Endnutzern den Umgang mit neuen Versionen erheblich. Um Anwender und Nutzer hinsichtlich der Ontologieevolution und Änderungen zu unterstützen, bedarf es automatisierter Werkzeuge und Analysetools,

[4] `http://www.ebay.com`
[5] `http://www.amazon.com`

welche eine Einschätzung und Erfassung der Evolution ermöglichen. Die Werkzeuge müssen zum einen in der Lage sein, Änderungen zwischen Ontologieversionen zu erkennen und zum anderen Funktionen bieten, welche die Evolution einer Ontologie quantitativ analysieren können.

Im nachfolgenden werden einige Desiderate und Anforderungen an eine effektive Unterstützung von Ontologieevolution diskutiert. Im Kapitel zu Verwandten Arbeiten (Kapitel 2) werden diese zum Vergleich existierender Systeme herangezogen. Ebenfalls findet eine Einordnung und Abgrenzung dieser Arbeit zu bereits existierenden Arbeiten statt (siehe 2.5). Eine umfassende Unterstützung von Ontologieevolution erfordert die Beachtung folgender Anforderungen:

- *Verfügbarkeit von einfachen und komplexen Änderungsoperationen:* Einfache Änderungen betreffen die Modifikation einzelner Ontologieelemente wie das Einfügen eines neuen Konzepts oder die Änderung eines Attributwertes. Hingegen beziehen sich komplexe Änderungsoperationen häufig auf mehrere Elemente, z. B. das Zusammenfassen mehrerer Konzepte zu einem Konzept (merge). Es existieren grundlegend zwei Möglichkeiten Änderungen zu spezifizieren. Einerseits können Änderungen explizit umgesetzt werden, d. h. eine Änderung wird direkt angewandt und die Ontologie wird somit inkrementell verändert. Andererseits kann auch nur die veränderte Ontologieversion O' vorliegen, d. h. die Änderungen werden implizit im Vergleich zur ursprünglichen Ontologieversion O spezifiziert.

- *Unterstützung für Mappings:* Für die automatische Propagierung von Änderungen in Instanzen oder andere Ontologien bzw. Mappings sollten die Änderungen explizit in einem sogenannten *Evolution-Mapping* erfasst werden. Im einfachsten Fall besteht ein solches Evolution-Mapping zwischen zwei Versionen O und O' aus dem durch den Entwickler umgesetzten Änderungsoperationen. Ist lediglich die veränderte Ontologieversion O' gegeben, muss das Evolution-Mapping durch eine gesonderte Differenzbestimmung erstellt werden. Diese kann auf Basis eines Schema-/Ontologie-Matching [104] erfolgen. Zusätzlich müssen ebenfalls eingefügte bzw. gelöschte Ontologieelemente, welche nur in einer der beiden Versionen vorliegen, beachtet werden.

- *Propagierung von Änderungen in abhängige Ontologien und Mappings sowie Instanzen:* Ontologien werden oftmals z. B. über Annotation- oder Ontologie-Mappings mit anderen Datenquellen bzw. Ontologien verknüpft. Aufgrund der Abhängigkeiten sollten Änderungen an Ontologien auch entsprechend in abhängige Mappings usw. propagiert werden, damit diese weiterhin konsistent und aktuell erscheinen. Dieser Prozess sollte vorzugsweise zu einem Großteil automatisch ablaufen, d. h. ohne aufwendige manuelle Eingriffe eines Anwenders. Gleiches gilt auch für die Migration von Instanzen einer Ontologie.

- *Unterstützung von Versionierung:* Ontologieevolution sollte zusätzlich durch

ein Versionierungssystem unterstützt werden, welches einen transparenten Zugang zu verschiedenen Versionen einer Ontologie anbietet. Somit können beispielsweise Anwendungen weiterhin auf eine bestimmte Version zurückgreifen obwohl bereits neuere Versionen einer Ontologie existieren, d. h. eine Anwendung muss nicht direkt an die neueste Version angepasst werden. Gleichermaßen können Fehlentwicklungen während der Evolution rückgängig gemacht werden, indem die aktuelle Version einfach durch eine ältere, korrekte Version ersetzt wird. Es sind verschiedene Ansätze denkbar, z. B. eine sequentielle Versionierung in der jede Version exakt eine Vorgänger- bzw. Nachfolgeversion besitzt oder eine parallele Versionierung.

- *Infrastruktur:* Die zuvor besprochenen Funktionalitäten sollten in einer mächtigen Infrastruktur bestehend aus einfach verwendbaren Werkzeugen verfügbar sein. So sollte es beispielsweise möglich sein das Ausmaß von Änderungen bestimmen zu können sowie inkrementell Änderungen zu spezifizieren. Weiterhin sollte die Bestimmung von Evolution-Mappings und Versionierung unterstützt werden. Benutzerfreundliche User-Interfaces zur Umsetzung von Änderungen bzw. zur Analyse der Ontologieevolution runden eine komplette Infrastruktur ab.

Innerhalb dieser Arbeit werden verschiedene Probleme der Evolution von Ontologien in den Lebenswissenschaften thematisiert, welche im folgenden Nutzerszenario illustriert werden. Möchte ein Nutzer beispielsweise eine Ontologie langfristig in einem seiner Projekte inkl. Analysen und Anwendungen einsetzen, interessiert ihn vor allem die vergangene Entwicklung der Ontologie, z. B. Fragen wie „Ist die Ontologie abgeschlossen oder befindet sie sich noch in Entwicklung?“ oder „Wie viele Konzepte wurden in den letzten drei Jahren eingefügt oder gelöscht?“. Darüber hinaus interessieren ihn Detailänderungen an bestimmten Konzepten, welche er für seine Analysen als wichtig erachtet. Bei Veröffentlichungen neuer Ontologieversionen möchte er wissen, welche Änderungen sich gegenüber seiner zuletzt verwendeten Version ergeben haben. Das Ziel ist daher ein möglichst automatisches Verfahren zu besitzen, welches genau diese Änderungen, d. h. den Diff zwischen den beiden Ontologieversionen bestimmen kann. Da er an bestimmten Teilgebieten der Ontologie interessiert ist, möchte er zudem wissen, ob seine Gebiete besonders starken Änderungen unterlagen, d. h. instabil sind oder ob sie als stabil (unverändert) gelten. Somit kann sich der Nutzer ebenfalls über neue bzw. aufstrebende Gebiete innerhalb der Ontologie informieren und diese zeitnah in seinen Applikationen verwenden. Aus dem dargestellten Nutzerszenario lassen sich die beiden folgenden in dieser Arbeit behandelten Themen ableiten:

Änderungserkennung in Ontologien: Die Erkennung von Änderungen in Ontologien z. B. zwischen zwei Ontologieversionen O_{v1} und O_{v2} bildet die Grundlage für den Umgang mit Ontologieevolution. So möchten Anwender von Ontologien bei der Veröffentlichung neuer Ontologieversionen in erster Linie wissen,

was sich an der Ontologie verändert hat. Erst dann können sie weitere Schritte wie die Planung von Anpassungen bzw. Umstrukturierungen angehen. Im günstigsten Fall haben die angefallenen Änderungen keine Auswirkungen, wodurch die neue Version O_{v2} problemlos die alte Version O_{v1} ersetzen kann. Eine manuelle Bestimmung der Änderungen ist aufgrund der Größe der Ontologien unrealistisch, daher wurden in der Literatur erste Verfahren zur automatischen Änderungserkennung vorgeschlagen. Ziel ist es die möglichst vollständige Menge von Änderungen zu bestimmen, wobei nicht nur einfache Änderungsoperationen wie Einfügungen oder Löschungen sondern auch komplexe Änderungen wie das Zusammenfassen oder Aufspalten von Konzepten erkannt werden sollen. Durch die zunehmende Größe der Ontologien verlieren Nutzer jedoch oftmals den Überblick über die angefallenen Änderungen. Zudem sind sie meist nur an bestimmten Sachverhalten interessiert und verwenden daher nur bestimmte Ontologieteile (Gebiete einer Ontologie). Deshalb besteht ein gesteigertes Interesse zu wissen, welche Ontologieteile besonders änderungsintensiv sind oder welche kaum Änderungen aufweisen.

Quantitative Evolutionsanalysen: Während das Thema Änderungserkennung primär die Bestimmung der Änderungen in Ontologien verfolgt, hat die quantitative Evolutionsanalyse von Ontologien das Ziel die angefallenen Änderungen quantitativ zu beurteilen und die Entwicklung (Evolution) einer gesamten Ontologie einzuschätzen. So werden sich Anwender, welche eine neue Ontologie einsetzen wollen, neben den typischen Fragestellungen wie z. B. „Passt die Ontologie zu meiner Anwendung?“ ebenfalls für Fragen zur Entwicklung und Evolution der Ontologie interessieren, z. B. „Wie stark ist die Ontologie in den letzten X Jahren angewachsen?“, „Welche Änderungen dominieren die Evolution?“ oder „Wie hat sich die Struktur der Ontologie verändert?“. Um diese Fragen beantworten zu können, bedarf es einer quantitativen Analyse der Evolution in Ontologien über längere Zeiträume hinweg. Das Ziel ist es dabei, dem Nutzer mittels geeigneter Maße Informationen über die Evolution der entsprechenden Ontologien zur Verfügung zu stellen. Auf Basis dieser Informationen können dann weitere Planungen oder Entscheidungen bzgl. des Umgangs mit der Ontologie erfolgen.

Die genannten Themengebiete werden in den beiden Teilen – Algorithmen zur Änderungsbestimmung – sowie – Systeme zur Analyse der Ontologieevolution – behandelt. Eine Diskussion verwandter Arbeiten und Systeme erfolgt in Kapitel 2.

1.2 Wissenschaftlicher Beitrag

Der wissenschaftliche Beitrag dieser Dissertation besteht aus den folgenden drei Arbeiten auf dem Gebiet Evolution von Ontologien in den Lebenswissenschaften:

Framework für Evolutionsanalysen von Ontologien und Mappings: Das Framework ermöglicht gleichzeitig quantitative Evolutionsanalysen für Ontologien und Ontologie-basierte Mappings wie Annotationen oder Ontologie-Mappings. Dazu stellt das Framework verschiedene Metriken zur quantitativen Beurteilung der Evolution bereit. Die darauf basierte Webanwendung OnEX (Ontology Evolution Explorer) erlaubt eine ad-hoc Analyse der Evolution von Ontologien sowie die Exploration von Änderungen an einzelnen Ontologieelementen. Dies ermöglicht Nutzern von Ontologien eine Einschätzung der Evolution und kann somit als Hilfestellung für den Umgang mit sich ständig verändernden Ontologien dienen. Der zur Evolutionsanalyse verwendete Versionierungsansatz gestattet eine effiziente Verwaltung mehrerer Versionen verschiedener Ontologien. Die Skalierbarkeit des Ansatzes erlaubt die Versionierung großer Ontologien mit einer hohen Anzahl zu analysierender Versionen.

Diff von Ontologieversionen: Es wird ein regelbasierter Ansatz zur automatischen Bestimmung von Evolution-Mappings zwischen Ontologieversionen vorgestellt. Dabei werden aufbauend auf einem Match zwischen zwei Ontologieversionen Regeln zur Erkennung von Änderungen angewandt. Neben der Bestimmung einfacher Änderungen wie dem Einfügen oder Löschen von Konzepten wird gleichermaßen die Erkennung komplexer Änderungsoperationen wie das Zusammenführen (merge) oder das Aufspalten von Konzepten (split) unterstützt. Die Erfassung des Diffs durch semantisch ausdrucksstarke Änderungsoperationen ermöglicht dabei eine kompakte und intuitive Repräsentation der Ontologieevolution. Es wird gezeigt, wie die berechneten Evolution-Mappings für die Migration von Ontologieversionen verwendet werden können. Der Diff Algorithmus wurde prototypisch implementiert. Die Evaluierung mittels großer Ontologien aus den Lebenswissenschaften und dem Web zeigte, dass der Ansatz in der Lage ist korrekte Evolution-Mappings effizient zu bestimmen. Die Adaptierbarkeit des Ansatzes gestattet eine flexible Anpassung an verschiedene Anwendungsszenarien und Ontologien.

Erkennung (in)stabiler Ontologieregionen: Es wird ein Verfahren zur automatischen Erkennung (in)stabiler Ontologieregionen, d. h. Ontologieteilen mit erhöhter bzw. geringer Änderungsintensität vorgeschlagen. Im Gegensatz zu anderen Ansätzen, welche vorrangig auf die Erkennung einzelner Änderungen fokussieren, steht hier die Bestimmung von Änderungen betroffener Ontologieregionen im Mittelpunkt. Dazu werden Ontologieregionen definiert und deren Stabilität über Metriken beurteilt. Im Zentrum steht ein Algorithmus, welcher für veröffentlichte Ontologieversionen innerhalb eines Zeitraums die (in)stabilsten Regionen bestimmen kann. Das Verfahren wurde anhand weit verbreiteter und großer Ontologien aus den Lebenswissenschaften evaluiert. Die Ergebnisse zeigen, dass das Verfahren in der Lage ist von Evolution betroffene Ontologieregionen zu klassifizieren. Die vorgestellte Trendanalyse ermög-

licht Nutzern eine Beurteilung der Ontologieevolution über längere Zeiträume hinweg und gestattet Designern die Planung weiterer Entwicklungen und Änderungen.

Die in der vorliegenden Arbeit dargestellten Ergebnisse wurden bereits als begutachtete Beiträge bei internationalen Konferenzen, Workshops, oder Journals publiziert.

Ein Übersichtsartikel zu aktuellen Arbeiten und Werkzeugen auf dem Gebiet der Schema- und Ontologieevolution wurde innerhalb des Buchbeitrags [53] im Buch [9] veröffentlicht. Erste Analyseergebnisse zur Evolution von Ontologien und Mappings in den Lebenswissenschaften wurden 2008 auf der internationalen DILS-Konferenz präsentiert [49]. Die Präsentation einer effizienten Versionsverwaltung für große Ontologien [60] erfolgte 2009 auf dem internationalen OntoContent-Workshop. Aufbauend darauf wurde die webbasierte Applikation OnEX (Ontology Evolution Explorer), welche 2009 in BMC Bioinformatics erschien [48], konzipiert und realisiert. Der Diff Algorithmus liegt in einer Preprint-Version vor [47] und befindet sich derzeit bei einem internationalen Journal unter Begutachtung. Auf der DILS-Konferenz 2010 wurde der Algorithmus zur Bestimmung und Erkennung von (in)stabilen Ontologieregionen [46] präsentiert. Weitere Arbeiten zur Beurteilung der Stabilität von Mappings wurden im Rahmen der BTW-Konferenz [116] sowie auf der internationalen DILS-Konferenz [38] in 2009 vorgestellt.

1.3 Aufbau der Arbeit

Die nachfolgende Arbeit gliedert sich in drei Teile. Im ersten Teil werden verwandte Arbeiten diskutiert und die Grundlagen beschrieben. Er umfasst die folgenden Kapitel:

Kapitel 2 gibt eine Einführung in die Thematik der Evolution von Schemas und Ontologien. Es erfolgt eine Diskussion relevanter Literatur zu Schema- und Ontologieevolution sowie Versionierung und Änderungserkennung. Abschließend wird die vorliegende Arbeit mit bisherigen Arbeiten auf dem Gebiet der Ontologieevolution verglichen.

Kapitel 3 vermittelt Grundlagen zu in der Arbeit verwendeten Modellen und Begriffen. Zunächst wird das verwendete Ontologiemodell definiert sowie die Versionierung von Ontologien beschrieben. Abschließend wird der Begriff des Mappings erläutert und die verschiedenen Mappingarten, Annotation-Mapping sowie Ontologie-Mapping, eingeführt.

Im zweiten Teil – Algorithmen zur Änderungsbestimmung – wird zunächst das zur quantitativen Evolutionsanalyse für Ontologien und Mappings konzipierte Frame-

work vorgestellt. Anschließend werden zwei Ansätze zur Bestimmung von Änderungen zwischen Ontologieversionen und der Erkennung (in)stabiler Ontologieteile präsentiert. Die Darstellung gliedert sich in die folgenden Kapitel:

Kapitel 4 beschreibt eine vergleichende Evolutionsanalyse für Ontologien und Mappings in den Lebenswissenschaften. Zunächst werden das Evolutionsmodell sowie entsprechende Metriken eingeführt. Die Evaluierung erfolgt an 16 repräsentativen Ontologien aus den Lebenswissenschaften sowie zugehöriger Annotation-Mappings und automatisch erstellter Ontologie-Mappings für den Zeitraum 2004–2008.

Kapitel 5 stellt einen regelbasierten Diff Algorithmus zur Bestimmung von Evolution-Mappings zwischen Ontologieversionen vor. Dabei werden zunächst mögliche Änderungsoperationen erläutert und der Begriff des Evolution-Mappings definiert. Anschließend erfolgt die Darstellung von Regeln zur Erkennung von Änderungen. Im Hauptteil wird der auf den Regeln basierte Algorithmus erläutert. Im Unterschied zu bisherigen Ansätzen gestattet die Verwendung von Regeln eine flexible Bestimmung einfacher sowie komplexer Änderungsoperationen für verschiedene Ontologien. Als ein mögliches Anwendungsszenario wird auf die Migration von Ontologieversionen eingegangen. Abschließend werden Evaluierungsergebnisse des Algorithmus dargestellt.

Kapitel 6 beschreibt einen neuartigen Ansatz zur Bestimmung (in)stabiler Ontologieteile (Regionen), welche aufgrund der Ontologieevolution intensiven oder nur marginalen Änderungen unterlagen. Zunächst werden der Begriff der Ontologieregion und Metriken zur Stabilitätsbewertung einer Ontologieregion eingeführt. Der Schwerpunkt liegt auf der Darstellung des Algorithmus zur Bestimmung der (in)stabilen Regionen. Abschließend werden Evaluierungsergebnisse des Algorithmus präsentiert.

Im dritten Teil – Systeme zur Analyse der Ontologieevolution – wird zunächst das Online-Analysesystem OnEX vorgestellt. Anschließend wird auf die Versionierung von Ontologien eingegangen. Der Teil umfasst folgende Kapitel:

Kapitel 7 stellt die webbasierte Applikation OnEX (Ontology Evolution Explorer) vor. OnEX ermöglicht die Exploration von Änderungen sowie eine Einschätzung der Evolution von Ontologien mittels dreier Workflows. Es werden die Workflows Quantitative Evolutionsanalyse, Konzept-basierte Evolutionsanalyse sowie Annotation-Migration vorgestellt. Es erfolgt ebenfalls ein Einblick in die Architektur und technische Umsetzung der Webapplikation.

Kapitel 8 gibt einen Einblick in die für die Analysen und OnEX verwendete Versionsverwaltung von Ontologien. Es wird ein Modell zur effizienten Verwaltung

von Ontologieversionen vorgestellt. Der Schwerpunkt liegt auf der Beschreibung eines Repositories zur Versionsverwaltung sowie dem Import von Ontologieversionen. In der Evaluierung wird die Funktionalität und Effizienz des Versionierungsansatzes gezeigt.

Nach einer Zusammenfassung der Ergebnisse der Dissertation sowie einem Ausblick auf zukünftige Arbeiten werden im Anhang Details zu einzelnen Arbeiten präsentiert. So werden für den Diff Algorithmus alle definierten Änderungsoperationen sowie die verwendeten Regeln aufgelistet. Ebenfalls können Detailstatistiken und Diagramme der Evolutionsanalysen eingesehen werden.

2

Verwandte Arbeiten – Schema- und Ontologieevolution

In diesem Kapitel wird ein Überblick über verwandte Arbeiten auf dem Gebiet der Schema- und Ontologieevolution gegeben. Nach einer Einführung (2.1) werden im zweiten Teil Forschungsarbeiten bzw. Systeme in den Gebieten der relationalen Schemaevolution (2.2), XML Schemaevolution (2.3) und Ontologieevolution (2.4) vorgestellt. Abschließend erfolgt eine Zusammenfassung und Abgrenzung der vorliegenden Arbeit (2.5).

2.1 Einführung

Die Evolution von Schemas und die damit verbundenen Probleme sind ein andauerndes Forschungsthema. Im Allgemeinen bezeichnet Schemaevolution die Fähigkeit bereits in Verwendung befindliche Schemas abzuändern bzw. anzupassen. Unter dem Begriff Schema werden verschiedene strukturierte Metadaten zur Beschreibung von komplexen Strukturen wie Datenbanken, Nachrichten, Applikationen oder Workflows verstanden. Beispiele für Schemas sind relationale Datenbankschemas, ER- oder UML-Modelle, Ontologien, XML Schemas, Softwareschnittstellen oder Workflowbeschreibungen. Die Gründe für die Evolution eines Schemas sind vielfältig. So besteht in der Regel ein Bedarf an Schemaevolution wann immer die Designer oder Entwickler eines Schemas neue oder veränderte Anforderungen umsetzen müssen. Beispiele sind u. a. die Behebung von Fehlern in einem aktuellen Schema, die Ein-

arbeitung von neuem Domänenwissen oder die Migration zu einer neuen Plattform.

Eine umfassende Unterstützung von Schemaevolution ist schwierig, da Schemaänderungen häufig in die von einem Schema abhängigen Strukturen/Komponenten, z. B. Daten (Instanzen), Sichten oder Applikationen, propagiert werden müssen. Eine ideale Unterstützung sollte dabei den manuellen Aufwand bei der Umsetzung von Schemaänderungen so weit wie möglich reduzieren, d. h. automatische Verfahren sollten den Entwickler bei der Durchführung von Evolution unterstützen. Beispielsweise sollten Änderungen an einem relationalen Schema einer Datenbank zu den entsprechenden Instanzdaten bzw. Sichten ohne allzu hohen manuellen Aufwand propagiert werden. Wird umgekehrt keine ausreichende Unterstützung für Schemaevolution geboten, gestaltet sich die Umsetzung von Änderungen sehr schwierig und daher zeitaufwendig, was unter Umständen zu Systemausfällen oder Fehlverhalten in den Anwendungen führen kann.

Schemaevolution ist seit geraumer Zeit ein aktives Forschungsfeld und wird auch zunehmend in kommerziellen Systemen unterstützt. Der Bedarf an einer leistungsstarken Unterstützung der Schemaevolution ist jedoch weiterhin ansteigend. Ein Grund hierfür ist die in den letzten Jahren gestiegene und weit verbreitete Nutzung neuartiger Schemas wie XML Schema, Web Services oder Ontologien, welche ebenfalls eine umfassende Unterstützung für Schemaevolution benötigen. Da die vorliegende Dissertation sich vorrangig mit der Evolution von Ontologien in den Lebenswissenschaften beschäftigt, werden nachfolgend die Arbeiten zur Evolution von relationalen Schemas (2.2) und XML Schemaevolution (2.3) im Vergleich zur Ontologieevolution (2.4) in einem kleineren Umfang diskutiert. Für die komplette Ausführung wird auf [53] verwiesen. In der Online-Bibliographie unter `http://se-pubs.dbs.uni-leipzig.de` [105] ist eine umfassende Sammlung verwandter Arbeiten zum Thema Schemaevolution abrufbar.

2.2 Schemaevolution in relationalen Datenbanken

Das wohl am häufigsten genutzte Modell zur Verwaltung von Daten ist das relationale Modell. Auf Basis von Relationen liegt eine feste Kopplung zwischen Schema und Instanzen vor, wobei Instanzen fest die Vorgaben des Schemas erfüllen müssen. Als Konsequenz aus dieser festen Bindung müssen Instanzen jeglicher Änderung des Schemas folgen, d. h. sie müssen angepasst werden um weiterhin valide gegenüber dem geänderten Schema zu sein. Dies betrifft ebenfalls die Anpassung von Bedingungen (Constraints) in einem Schema.

Eine weitere Problematik ist die häufig vorliegende enge Kopplung zwischen dem relationalen Schema einer Datenbank und abhängigen Applikationen. So wird beispielsweise häufig die Anfragesprache SQL in Anwendungen eingesetzt, um mit einer zugehörigen Datenbank zu interagieren (z. B. für die Abfrage von Daten oder

das Sichern von Daten). Nach einer Änderung des Schemas versuchen nun Applikationen weiterhin auf die Datenbank zuzugreifen. Eine Unterstützung seitens SQL bilden sogenannte Sichten (Views). Möchte eine Applikation neue Anforderungen umsetzen, bestehen verschiedene Möglichkeiten. Einerseits können Sichten angelegt werden, um die neue Version der Applikation zu unterstützen, wobei existierende Strukturen für ältere Versionen belassen werden. Andererseits kann auch das Schema selbst verändert werden, um die neuen Anforderungen umzusetzen. In diesem Fall müssen evtl. existierende Sichten für ältere Versionen angepasst werden, um eine Rückwärtskompatibilität für andere Anwendungen zu gewährleisten.

Im Folgenden wird zunächst ein Überblick über die Unterstützung von Schemaevolution in kommerziellen Datenbankmanagementsystemen gegeben. Anschließend erfolgt eine Darstellung von Forschungsarbeiten für die Unterstützung von relationaler Schemaevolution.

2.2.1 Kommerzielle Datenbanksysteme

Relationale Datenbanksysteme bieten SQL Anweisungen wie CREATE, DROP oder ALTER um Schemaänderungen durchzuführen. Die konkreten Umsetzungen können jedoch von System zu System variieren [117]. So kann eine neue Spalte C mit dem Datentyp Integer mit folgender Anweisungen in eine Tabelle T eingefügt werden:

```
ALTER TABLE T ADD COLUMN C int;
```

Andere Änderungen werden je nach System verschieden umgesetzt. So ist die Umbenennung einer Tabelle in Oracle wie folgt möglich:

```
ALTER TABLE foo RENAME TO bar;
```

In SQL Server wird mittels einer Stored Procedure die Umbenennung ermöglicht:

```
sp_rename 'foo','bar','TABLE';
```

Die in SQL verfügbaren Anweisungen für Schemaänderungen sind durchweg atomar, d. h. ohne eine Erweiterung beschreibt jede Anweisung jeweils eine einfache Änderung am Schema. So können individuelle Tabellen eingefügt oder gelöscht werden, ebenso Spalten oder Constraints. In Ergänzung können auch bestehende Schemaelemente geändert werden, z. B. die Umbenennung einer Spalte oder Eigenschaften einer Spalte wie Datentyp, maximale Länge oder Präzision. Komplexe Änderungen wie das horizontale oder vertikale Splitten einer Tabelle oder das Zusammenlegen (merge) von Tabellen werden in kommerziellen Systemen nicht unterstützt. Solche Änderungen müssen mit Hilfe einer sequentiellen Ausführung mehrerer atomarer Änderungen realisiert werden. Dies ist immer möglich, jedoch bleibt die eigentliche Intention (Semantik) der komplexen Änderung hinter der Summe der einfachen Änderungen verborgen, z. B. dass eine Tabelle in zwei Tabellen aufgesplittet wurde.

Kommerzielle Systeme führen eine automatische Update-Propagierung für einfache

Änderungen durch. So führen einfache Änderungen wie das Einfügen/Löschen einer Spalte oder die Änderung des Datentyps zu einer direkten Anpassung der entsprechenden Instanzen. Des Weiteren ist die direkte (online) Reorganisation ein wichtiger Bestandteil von Datenbanksystemen, um Ausfallzeiten von Servern für die Zeit der Updates zu vermeiden. Auf der anderen Seite gibt es nur wenig Unterstützung für die Propagierung von Schemaänderungen auf abhängige Schemaelemente wie Sichten, Indexe oder Fremdschlüssel. Entweder müssen betroffene Elemente manuell angepasst oder die gesamte Schemaänderung muss unter Umständen zurück genommen werden.

Kommerzielle Systeme unterstützen primär keine abstrakten Schema-Mappings sondern lediglich SQL zur Definition von Sichten oder Evolution-Mappings. Es existiert keine Unterstützung für explizite Schema- und Datenbankversionen. Sobald eine Änderung spezifiziert in SQL umgesetzt wurde, geht der vorherige Status der geänderten Elemente verloren. Weiterhin gibt es keine Unterstützung für Anwendungen, welche auf einer älteren Version der Datenbank entwickelt wurden. Nachfolgend wird auf Spezifika einzelner Anbieter näher eingegangen.

Oracle stellt ein Werkzeug namens *Change Management Pack* [94] bereit, welches die Ausführung einiger komplexer Schemaänderungen ermöglicht. Das Werkzeug erlaubt den Vergleich zweier Datenbankschema und bestimmt, ob mögliche Einflüsse oder Fehler aufgelöst werden müssen. Seit Version 9i wird das Schemaevolution-Werkzeug *redefinition* angeboten [92]. Redefinition ermöglicht Administratoren die Spezifikation und Ausführung mehrerer Schemaänderungen an einer Tabelle. Änderungen können durchgeführt werden, wobei die betreffende Tabelle weiterhin Anwendungen zur Verfügung steht (d. h. es gibt keine Ausfallzeiten). Ein weiterer Bestandteil der Schemaevolution in Oracle bilden *editions* [93]. Eine Edition ist eine logische Gruppierung von Datenbank- und Schemaelementen wie Sichten oder Trigger, welche Applikationen für Datenbankzugriffe zur Verfügung stehen. Somit kann eine Edition als eine gekapselte Version einer Datenbank angesehen werden.

SQL Server stellt Nutzern ein Werkzeug namens *SQL Server Management Studio (SSMS)* [81] bereit. SSMS verfügt über ein graphisches Nutzerinterface womit Änderungen an einer Datenbank, z. B. das Einfügen von Fremdschlüsseln oder das Löschen von Spalten auf Basis eines visuellen Diagramms durchgeführt werden können. SQL Server bietet weiterhin ein Werkzeug namens *Data-Tier Applications* an [80]. Im Mittelpunkt steht dabei eine verteilbare Datei *DAC pack*. Ein DAC pack ist nichts anderes als ein einsetzbares Abbild einer Version eines Datenbankschemas für Applikationen. Typischerweise wird eine Version einer Applikation inklusive ihres zugehörigen Datenbankschemas innerhalb eines DAC pack gekapselt.

IBM DB2 enthält ein Werkzeug *Optim Data Studio Administrator* zur Anzeige, Erzeugung und Veränderung von Datenbankobjekten [57]. Es können einerseits Änderungen einzeln und manuell abgesetzt werden. Andererseits erlaubt ein Batch-Modus die Zusammenfassung von Änderungen innerhalb eines Skripts, welches statisch auf

Fehler und Ausführbarkeit der Änderungen geprüft wird.

2.2.2 Forschungsansätze

PRISM [21, 23] ist ein Prototyp innerhalb des Projektes *Panta Rhei*, welches Werkzeuge zur Schemaevolution untersucht und erforscht. PRISM zur Unterstützung der Schemaevolution in relationalen Datenbanken fokussiert dabei auf zwei Hauptziele: (1) Durchführung von Schemaevolution mit einer verbesserten Semantik und Datenerhaltung sowie (2) die Unterstützung mehrerer Versionen einer Applikation, um auf ein und denselben Daten arbeiten zu können. Ein Beitrag von PRISM ist eine Sprache zur Manipulation von Schemaelementen mit Hilfe sogenannter *Schema Modification Operators (SMOs)*. Die Sprache ähnelt sehr dem SQL Dialekt, d. h. sie ist ebenfalls deklarativ und enthält auch einige bekannte Anweisungen wie „CREATE TABLE“ oder „ADD COLUMN“. Es bestehen jedoch zwei grundlegende Unterschiede.

Erstens wird für jede SMO Anweisung eine formale Semantik erfasst, welche die Evolution in Vorwärts- bzw. Rückwärtsrichtung an Schemas beschreibt. Dabei beinhaltet die Rückrichtung die Umkehraktionen, welche durchgeführt werden müssen, um die Evolution in Vorwärtsrichtung rückgängig zu machen. Zweitens besteht zwischen SMO und SQL ein Unterschied bzgl. der Definition einer atomaren Änderung. SQL besitzt eine Abschlusseigenschaft, d. h. jeder kann ein Schema S in ein anderes Schema S' mit Hilfe einer Sequenz von Anweisungen überführen. Evtl. tritt dabei ein Datenverlust auf, aber die Ausführung einer solchen Sequenz sollte immer möglich sein. Anweisungen in SMO verfolgen eine andere Philosophie. Jede Anweisung stellt dabei eine Reorganisation der Datenbank verbunden mit einer Datenmigration dar. Die verfügbaren Anweisungen in SMO fokussieren dabei auf Änderungen zur Reorganisation einer Datenbank auf höherer Ebene als SQL. Beispiele für Änderungen sind die folgenden:

```
MERGE TABLE R, S INTO T
```

```
PARTITION TABLE T INTO S WITH T.X < 10, T
```

```
COPY TABLE R INTO T
```

Die drei Anweisungen fassen zwei Tabellen zusammen (MERGE), Splitten eine Tabelle auf Basis eines Prädikats auf (PARTITION) oder Kopieren den Inhalt einer Tabelle (COPY). Jede Anweisung und dessen Inverse kann mit Hilfe von logischen Formeln sowie einer Sequenz von SQL Anweisungen beschrieben werden. So würde obige MERGE Anweisung mit folgender SQL Sequenz korrespondieren:

```
CREATE TABLE T (columns from R or S)
```

```
INSERT INTO T
        SELECT * FROM R UNION SELECT * FROM S
```

```
DROP TABLE R

DROP TABLE S
```

Der zweite Beitrag von PRISM fokussiert auf die Versionierung mit Vorwärts- und Rückwärtskompatibilität. Bei Erzeugung einer neuen Version N+1 auf Basis der Version N mit Hilfe von SMO Anweisungen bietet PRISM die folgenden Unterstützungen für Anwendungen an:

- Automatische Transformation von Anfragen auf Basis der Version N in semantisch äquivalente Anfragen gegen das Schema in Version N+1 oder umgekehrt
- Erstellung von Sichten für Version N, welche auf Version N+1 basieren

PRISM wurde anhand der Änderungshistorie des Mediawiki-Schemas [8], welches zur Verwaltung der Inhalte der Wikipedia verwendet wird, evaluiert [22]. Dabei wurde eine Klassifikation von High-Level Änderungen vorgenommen, welche einen Großteil der angefallenen Änderungen in der Historie des Repositories abdeckt.

HECATAEUS [97] fokussiert auf Abhängigkeiten zwischen Schemaelementen und abhängigen Strukturen wie Sichten oder Anfragen. Kommerzielle Systeme (siehe 2.2.1) haben starke Restriktionen bzgl. Schemaevolution wenn Abhängigkeiten vorliegen, z. B. darf eine Spalte nicht gelöscht werden falls diese in einem Index referenziert wird. HECATAEUS stellt diesbezüglich eine verbesserte Kontrolle bereit, d. h. Administratoren können entscheiden wann Änderungen in abhängige Strukturen wie Instanzen, Sichten oder Anfragen propagiert werden oder nicht. Das zentrale Konstrukt in HECATAEUS sind *evolution policies* [96]. Policies können beispielsweise bei der Erstellung von Tabellen, Sichten, Constraints oder Anfragen in SQL angegeben werden. Ein Beispiel für eine Policy stellt die folgende Erstellung einer Tabelle dar:

```
CREATE TABLE Person (

Id INT PRIMARY KEY,

Name VARCHAR(50),

DateOfBirth DATE,

Address VARCHAR(100),

ON ADD Attribute TO Person THEN Propagate)
```

Die Anweisung gibt an, dass das Hinzufügen eines Attributs (Spalte) automatisch in abhängige Strukturen propagiert werden soll, d. h. eine neue Spalte würde dann ebenfalls in der abhängigen Struktur eingefügt. Als ein Beispiel soll die nachfolgende Sicht auf die Tabelle 'Person' dienen:

```
CREATE VIEW BobPeople AS
```

```
SELECT Id, DateOfBirth, Address FROM Person
WHERE Name = 'Bob'
```

Wird beispielsweise in 'Person' eine neue Spalte 'City' eingefügt, so würde die 'Bob-People' Sicht automatisch um die gleiche Spalte ergänzt werden. Policies werden stets auf den Elementen, welche geändert werden sollen, definiert. Es werden drei Typen von Policies unterschieden. Die Option *propagate* propagiert automatisch alle Änderungen in die abhängigen Strukturen (siehe obiges Beispiel). Mit der Option *block* kann die Propagierung in abhängige Strukturen verweigert werden. Die *prompt* Option gestattet Nutzern eine manuelle Entscheidung darüber, ob propagiert werden soll oder nicht.

DB-MAIN [45] ist eine Plattform zur Modellierung von Datenbanken auf Basis von konzeptionellen Modellen. Nutzer können u. a. eine Reverse-Engineering durchführen, um ein ER-Modell einer bestehenden Datenbank zu erzeugen. Die Abbildung (das Mapping) zwischen Modell und Datenbankschema ist relativ einfach gehalten. So werden Konstrukte im konzeptionellen Modell, welche kein direktes Pendant im Datenbankschema besitzen (z. B. Vererbungsbeziehungen), auf Pattern wie beispielsweise Fremdschlüssel abgebildet. Dadurch ist auch ein Reverse-Engineering auf Basis der Detektierung vordefinierter Pattern möglich. DB-MAIN unterstützt Schemaevolution indem Änderungen am konzeptionellen Modell oder am Datenbankschema jeweils entsprechend propagiert werden (Synchronisation zwischen konzeptionellen Modell und Datenbankschema) [55]. So werden beispielsweise Änderungen am konzeptionellen Modell auf das Datenbankschema propagiert ohne dass die darunterliegende Datenbank komplett neu aufgebaut werden muss. Die Änderungen am Modell werden in entsprechende SQL Anweisungen übersetzt, welche zur Anpassung des Datenbankschemas ausgeführt werden. Eine Garantie für Rückwärtskompatibilität wird nicht gegeben.

MeDEA [28] ist ein ähnliches Werkzeug wie DB-MAIN. Der Hauptunterschied zu DB-MAIN besteht darin, dass MeDEA weder eine feste Sprache zur Beschreibung der Evolution noch ein fixes Mapping zwischen konzeptionellen Modell und Datenbankschema besitzt. So kann beispielsweise das konzeptionelle Modell in UML oder erweitertem ER vorliegen. Aufgrund dessen existieren für die Abbildung zwischen konzeptionellen Modell und Datenbankschema mehrere Möglichkeiten, d. h. ein Konstrukt im konzeptionellen Modell kann auf verschiedene Art und Weise im Datenbankschema umgesetzt werden. Das Schlüsselkonzept von MeDEA sieht dafür sogenannte *evolution choices* in Form von Regeln vor. Der Entwickler wählt bei jeder Änderung eine passende Regel aus, welche dann das Datenbankschema entsprechend anpasst. Beispielsweise existieren bei einem ER-Modell für die Einfügung einer neuen Entität, welche von einer bestehender Entität ableitet, die folgenden Optionen zur Anpassung des Datenbankschemas:

- Einfügen einer neuen Tabelle mit dem Primärschlüssel der Hierarchie und den

neuen Attributen als Spalten inklusive eines Fremdschlüssels zur Vatertabelle („table-per-type“ Strategie)

- Einfügen einer neuen Tabelle mit Spalten, welche alle neuen Attribute sowie die vererbten Attribute abdecken („table-per-concrete-class“ Strategie)
- Einfügen von zusätzlichen Spalten in die Tabelle der Vaterentität wobei ein Diskriminator eingefügt oder ein existierender Diskriminator zur Unterscheidung der Tupel verwendet wird („table-per-hierarchy“ Strategie)

Impact Analysis [74] ist ein Ansatz, welcher versucht die lose Kopplung zwischen Anwendungen und Schema zu überbrücken. Die Idee besteht darin Entwickler von Anwendungen über mögliche Effekte von Schemaänderungen während der Designphase zu informieren. Problematisch ist dabei die Tatsache, dass Applikationen häufig keine festen Anfragen an die Datenbank stellen, sondern diese dynamisch innerhalb der Applikation erzeugen. Ein Analysewerkzeug bestimmt, welche Anfragen durch die Applikation erzeugt werden und wertet gleichzeitig den Status der Applikation zum Zeitpunkt der Anfrage aus. Unter der Annahme von SQL-basierten Schemaänderungen werden mögliche Einflüsse auf Basis bekannter Refactoring-Techniken [2] kategorisiert. Beispielsweise führt die Löschung einer Spalte zu Fehlern in Anfragen, d. h. entsprechende Anfragen sind nicht mehr ausführbar („error-level impact“). Die Analyse versucht solche Situationen schon während der Designphase aufzudecken, damit Fehler nicht erst zur Laufzeit auftreten und bereits im Vorfeld behoben werden können. Andererseits wird z. B. bei der Spezifikation eines Default-Wertes für eine Spalte lediglich eine Warnung registriert („warning-level impact“), da die Anfragen weiterhin funktionieren jedoch die veränderte Semantik (Default anstatt NULL Werte) beachtet werden sollte.

Eine Vielzahl von Forschungsarbeiten wurde ebenfalls im Rahmen der Adaptierung von Mappings (mapping adapation) zur Unterstützung von Schemaevolution publiziert [118, 124]. Ein Überblick über relevante Arbeiten ist in [31] zu finden. Die Arbeiten basieren auf relationalen oder genesteten relationalen Schemas und verwenden verschiedene Arten logischer Schema-Mappings. Auf dieser Basis wurden zwei wichtige Operatoren *compose* und *inverse* für Mappings definiert, implementiert und untersucht. Diese Operatoren passen in das Konzept des Model Management [12, 78], einem generischem Framework, welches die Manipulation von Schemas und Mappings mit Hilfe von High-Level Operatoren im Rahmen des Schemamanagement inklusive Schemaevolution [10] unterstützt.

2.3 XML Schemaevolution

XML als Datenmodell unterscheidet sich aufgrund seines semi-strukturierten Charakters stark vom relationalen Modell. XML-Instanzen müssen i. Allg. keinem fest

vorgegebenen Schema genügen. Einzige Bedingungen sind einige Kriterien zur Wohlgeformtheit, z. B. jedes Startelement muss ein zugehöriges Endelement besitzen oder Attribute müssen lokal eindeutige Namen aufweisen. Einzelne Elemente können strukturierten, komplett unstrukturierten Inhalt oder einen Mix aus beiden enthalten. Der ursprüngliche Zweck und das heute noch dominierende Anwendungsfeld von XML ist die Strukturierung von Dokumenten sowie die Nutzung als Kommunikationsmedium (Datenaustausch). Der Gebrauch als Speichermedium ist lediglich ein zweitrangiges Anwendungsgebiet.

Aufgrund der Neuheit von XML Schemas sind Probleme der XML Schemaevolution im Vergleich zu denen im Bereich relationaler Schemas noch wenig erforscht. Zwar haben sich in den letzten Jahren mit der Document Type Definition (DTD) und XML Schema [32] zwei Schemasprachen für XML herauskristallisiert, jedoch besitzt keine der Sprachen eine zu SQL äquivalente ALTER Anweisung zur inkrementellen Anpassung von Schemas. Die beiden Sprachen weisen verschiedene Fähigkeiten und Ausdrucksstärken auf, was letztendlich im Rahmen der Evolution beachtet werden muss. Zum aktuellen Zeitpunkt basieren Frameworks zur Unterstützung von XML Schemaevolution auf keiner einheitlichen Sprache, um inkrementelle Änderungen an XML Schemas zu spezifizieren. Das W3C veröffentlichte 2006 ein Dokument, in welchem typische Anwendungsfälle für die Evolution von XML Schemas diskutiert werden [120]. Das Dokument schlägt keine Sprache zur Handhabung der Evolution vor, es werden lediglich die Semantik und ein mögliches Verhalten für verschiedene Arten inkrementeller Änderungen diskutiert.

Eine zentrale Eigenschaft von Schemasprachen wie DTD oder XML Schema ist die Spezifikation der Repräsentation von Elementen in XML Dokumenten inklusive Eigenschaften wie Ordnung und Kardinalitäten. Änderungssprachen greifen diese Eigenschaften auf und schlagen entsprechende Änderungen vor, z. B. Änderung der Kardinalität eines Elements, Umordnung von Elementen, Umbenennung eines Elements oder das Löschen/Einfügen eines Elements in eine Sequenz. In [83] wurde eine Taxonomie mit möglichen inkrementellen Änderungen für XML Schemas vorgeschlagen:

1. Einfügen eines neuen (optionalen) Elements in einen vorhandenen Typ
2. Löschen eines Elements aus einem vorhandenen Typ
3. Einfügen neuer top-level Konstrukte wie z. B. komplexe Typen
4. Löschen von top-level Konstrukten
5. Änderung der Semantik eines Elements ohne seine Syntax zu verändern, z. B. eine neue Version erfordert implizit die Angabe von Werten auf Basis des metrischen Einheitssystems

6. Modifikation eines Schemas ohne die Validität der Instanzen zu beeinflussen, z. B. Extraktion einer lokalen Typdefinition in eine einzelne globale Typdefinition
7. Nesten einer Kollektion von Elementen in einem anderen Element
8. Plätten eines Elements indem es durch all seine Kinder ersetzt wird
9. Umbenennung oder Anpassung des Namespace eines Elements
10. Änderung der minimalen bzw. maximalen Kardinalität eines Elements
11. Modifikation des Typs eines Elements, entweder durch Änderung eines benannten Typs oder durch Restriktion bzw. Erweiterung
12. Änderung des Default-Wertes eines Elements
13. Umordnung von Elementen innerhalb eines Typs

Für jede Klasse von Änderungen beschreibt [83] unter welchen Bedingungen eine Änderung die Vorwärts- bzw. Rückwärtskompatibilität erhält. Falls beispielsweise zwischen zwei Versionen $v1$ und $v2$ ein optionales Element X in einem Typ eingefügt wurde, so kann jede Applikation, welche auf $v1$ arbeitet ebenfalls mit $v2$ arbeiten und umgekehrt. Dies ist so lange möglich bis Applikationen Dokumente mit X Elementen erzeugen. Im Gegensatz dazu würde das Einfügen von X als Pflichtelement sofort zu einer Inkompatibilität der beiden Schemaversionen führen. Die gleichen Aussagen können ebenfalls für Instanzen getroffen werden, d. h. basiert ein XML Dokument auf Version $v1$ so ist dieses bei optionalen X Element auch gegenüber $v2$ valide, jedoch nicht wenn X ein Pflichtelement darstellt.

Anfang 2000 stellte DTD die dominierende Sprache zur Schematisierung von XML Dokumenten dar. Während der letzten Dekade hat sich das Bild geändert. Aktuell werden Schemas für XML Dokumente vorwiegend in XML Schema spezifiziert. Der gleiche Trend lässt sich in Arbeiten zur XML Schemaevolution beobachten. Ältere Arbeiten beziehen sich typischerweise auf das DTD Modell, wohingegen aktuelle Arbeiten auf XML Schema aufbauen. Nachfolgend werden Forschungsansätze im Bereich der XML Schemaevolution vorgestellt und diskutiert.

XEM (XML Evolution Management) [66, 115] ist ein in 2001 eingeführtes Framework zum Management der Evolution von DTDs. Das Framework basiert auf einer fundierten und vollständigen Menge von Änderungsoperationen. Jede vorhandene Änderungsoperation garantiert die Einhaltung aller Validitäts- und Integritätsbedingungen, d. h. nach Anwendung einer Änderungsoperation sind alle XML Dokumente weiterhin wohlgeformt und gültig bzgl. der geänderten DTD. Die Menge der Operationen ist komplett, d. h. jemand kann basierend auf einer DTD durch die sequentielle Anwendung von Operationen jedmögliche andere valide DTD konstruieren. Die Menge besteht aus den folgenden Schemaänderungen:

- Erzeugen eines DTD Elementtypen
- Löschen eines DTD Elementtypen
- Einfügen eines DTD Elements oder Attributs in einen existierenden Elementtypen
- Löschen eines DTD Elements oder Attributs aus einem existierenden Elementtypen
- Änderung der Kardinalität eines Elements in einem Elementtypen
- Nesten benachbarter Elemente in einem Typ in ein neues Element
- Plätten eines genesteten Elements

Jede Änderung an einer DTD führt i. Allg. zu Änderungen an abhängigen XML Dokumenten, um deren Validität gegenüber der DTD weiterhin zu gewährleisten. Erweitert beispielsweise jemand eine DTD um ein neues Pflichtelement, so fügt XEM automatisch entsprechende Default-Elemente in allen nicht-validen XML Dokumenten ein. Der Ansatz ähnelt demnach stark dem relationalen Schema, d. h. bei einer Schemaänderung werden abhängige Instanzen (XML Dokumente) entsprechend angepasst, um dem neuen (geänderten) Schema zu genügen.

DTD-Diff [70] ist ein Algorithmus zur Bestimmung von Änderungen zwischen zwei Versionen einer DTD. Die Eingabe besteht aus zwei DTD Schemaversionen für die eine Liste von Änderungen berechnet wird. Es werden die folgenden Änderungen erfasst:

- Einfügen oder Löschen eines Elements, Attributs oder Deklaration einer Entität
- Änderungen an Elementtypen wie das Einfügen, Löschen oder Umordnen von Subelementen
- Änderungen an der Kardinalität von Elementen
- Modifikationen an Attributen oder Eigenschaften von Entitäten wie die Änderung des Default-Wertes eines Attributs.

Aufgrund des voll-automatisierten Ansatzes wird eine Erkennung von Umbenennungen nicht explizit unterstützt.

Diagram-based Evolution [27] versucht über ein konzeptionelles Modell die Evolution von XML Schemas zu verwalten. Mit Hilfe eines Designwerkzeugs können Änderungen an einem konzeptionellen Modell durchgeführt werden, die Änderungen müssen entsprechend in die zugehörigen Schemas propagiert werden. Es werden

UML Diagramme als konzeptionelles Modell für XML Schema verwendet. Die UML Diagramme im Werkzeug unterstützen nicht die volle Mächtigkeit von XML Schema, aus diesem Grund fokussiert die Arbeit lediglich auf einen Teil von XML Schema, welcher unproblematisch in entsprechendes UML abgebildet werden kann. Die Änderungen am UML Modell führen zu Änderungen am darunterliegenden Schema und dessen Instanzen. Automatisch generierte XSLT Skripte werden zur Migration abhängiger XML Dokumente verwendet.

Ein ähnlicher Ansatz namens **CoDEX** [64] verwendet anstatt UML ein weitaus mehr an XML Schema angelehntes, konzeptionelles Modell. Wiederum werden inkrementelle Änderungen am konzeptionellen Modell in Änderungen am zugehörigen XML Schema und dessen Instanzen überführt. CoDEX stellt zusätzlich eine Algebra zur Verarbeitung der inkrementellen Änderungen bereit. Während der Nutzer sein Modell anpasst, werden die durchgeführten Änderungen geloggt. Anschließend kann mit Hilfe der Algebra und einiger Regeln eine Optimierung des Logs stattfinden. Beispielsweise kann die sequentielle Ausführung der Änderungen „Einfügen eines neuen Elements“ und anschließende „Umbenennung des Elements“ durch eine Änderung ersetzt werden: „Einfügung eines Elements mit dem neuen (geänderten) Namen“.

X-Evolution [41, 42, 43, 79] ist ebenfalls ein Framework zur inkrementellen Schemaevolution für XML Schema. Das Framework basiert auf einer graphartigen Repräsentation von XML Schema. Ähnlich wie CoDEX und andere UML Werkzeuge unterstützt X-Evolution ein Reihe von primitiven Änderungen. Des Weiteren werden auch XML Schema spezifische Änderungen angeboten, z. B. die Änderung einer Gruppe von ALL auf CHOICE oder SEQUENCE. Dabei werden einige der vorgeschlagenen Änderungen als unkritisch („no effect“) bzgl. der Validierung von XML Dokumenten eingestuft. Ein Beispiel hierfür ist das Löschen eines globalen Typs, welcher aktuell keine Elementinstanzen besitzt. Für diese Änderungen ist somit keine Revalidierung abhängiger XML Dokumente nötig. Ein wichtiger Beitrag von X-Evolution besteht in der inkrementellen Erkennung von invaliden Elementen und deren Revalidierung. Nach einer inkrementellen Änderung können Nutzer herausfinden, ob XML Dokumente immer noch valide gegenüber dem geänderten Schema sind. Falls dies nicht der Fall ist, besteht die Möglichkeit die XML Dokumente anpassen zu lassen, um mit dem neuen Schema valide zu sein. Anpassungen laufen „in-place“ ab, d. h. es werden keine komplett neuen Dokumente erzeugt, sondern nur die von Änderungen betroffenen Teile eines Dokuments werden angepasst.

Altova [1] stellt mit DiffDog ein Werkzeug zum Matching und Diff-Berechnung für XML Schemas bereit. Das Werkzeug erwartet zwei XML Schemas als Eingabe und führt darauf ein Element-Element Matching aus. Das Ergebnis des Match kann manuell nachbearbeitet werden, z. B. im Fall von Umbenennungen welche nicht automatisch erkannt worden sind. Auf Basis des Diff-Ergebnisses ist das Werkzeug in der Lage entsprechende XSLT Skripte zur Migration von XML Dokumenten zu generieren. Auf diese Art und Weise werden Änderungen wie Umbenennungen oder

das Umordnen von Elementen unterstützt. Es ist unklar wie mit Einfügungen von Pflichtelementen oder Änderungen in der Kardinalität von Elementen umgegangen wird. Es gibt zudem keinen Mechanismus zur inkrementellen Änderung der XML Schemas.

Forschung im Bereich Schema-Matching und Mapping brachte ebenfalls einige Werkzeuge zur semi-automatischen Bestimmung ausführbarer Mappings hervor, z. B. Clio zur Migration von Instanzen nach einer Schemaevolution [15, 59]. Die Werkzeuge fokussieren primär nicht auf eine inkrementelle Schemaevolution, jedoch wird ein Mapping zwischen dem alten und neuen Schema bestimmt. Die Werkzeuge unterstützen nur Teile von XML Schema, z. B. beherrscht Clio einen großen Teil von XML Schema, beachtet jedoch u. a. die Ordnung von Elementen oder CHOICE-Gruppen nicht.

2.4 Ontologieevolution

Gruber bezeichnet eine Ontologie als eine explizite Spezifikation einer Konzeptualisierung einer Domäne [39]. Obwohl eine Reihe verschiedener Arten von Ontologien existieren (siehe Einleitung), stellt eine Ontologie typischerweise ein gemeinsames/kontrolliertes Vokabular zur Modellierung einer Domäne bereit. Konzepte, welche miteinander über Beziehungen verknüpft sind und mittels Attributen wie Name oder Definition genauer charakterisiert werden, bilden die Entitäten der entsprechenden Domäne. In der jüngeren Vergangenheit wurden Ontologien immer stärker in verschiedenen Domänen verwendet, um beispielsweise die Eigenschaften von Objekten semantisch exakt zu erfassen oder Daten korrekt zu integrieren [112]. So ist gerade in den Lebenswissenschaften eine ständig anwachsende Zahl von Ontologien zu beobachten, z. B. die Ontologien welche innerhalb der Open Biomedical Ontologies (OBO) Initiative [110] oder im BioPortal [91] verwaltet werden. Die existierenden Ontologien sind nicht statisch, d. h. sie werden ständig angepasst und verändert, um das neueste Wissen der Domäne zu repräsentieren oder veränderte Anforderungen umzusetzen.

Zwischen Ontologien und relationalen Schemas existieren einige wichtige Unterschiede, welche bzgl. Evolution zu beachten sind:

- Ontologien sind aus konzeptioneller Sicht abstrakter als relationale Schemas und können in verschiedenen Varianten (basierend auf ihrer Ausdrucksstärke) wie kontrollierte Vokabulare und Thesauri, is_a-Hierarchien/Taxonomien, gerichtet azyklische Graphen oder frame-basierten bzw. formalen Repräsentationsformen auftreten [69]. So erlauben z. B. Ontologiesprachen wie RDF [73] oder OWL [76] u. a. die Spezifikation von Konzepthierarchien inklusive Mehrfachvererbung, Constraints und disjunkte Konzepte. Die Art und Ausdrucksstärke einer Ontologie bestimmt, welche Arten von Änderungen existieren.

Diese sollten im Rahmen einer Evolution unterstützt werden. Beispielsweise werden in [88] 22 einfache und komplexe Änderungsoperationen wie das Einfügen eines Konzepts, die Reklassifikation eines Konzepts oder das Splitten/Zusammenfassen von Konzepten vorgeschlagen.

- Die Rolle von Instanzen unterscheidet sich zwischen Ontologien und relationalen Schemas. So beinhalten viele Ontologien Instanzen, trennen diese jedoch nicht klar von den eigentlichen Konzepten der Ontologie. In anderen Fällen werden Instanzen durch Ontologiekonzepte näher beschrieben, dabei befinden sich die Instanzen jedoch in separaten Datenquellen, d. h. sie werden getrennt von der Ontologie verwaltet. Diese Unterschiede haben einen Einfluss auf die Propagierung von Ontologieänderungen, da separat verwaltete Instanzen typischerweise nicht unter der Kontrolle der Ontologieentwickler stehen.

- Im Gegensatz zu relationalen Schemas ist die Entwicklung und Evolution von Ontologien häufig ein kollaborativer und dezentraler Prozess. Des Weiteren werden neue Ontologien oft auf Basis bereits existierender Ontologien entwickelt, d. h. die Entwickler einer Ontologie nutzen eine bereits verfügbare Ontologie als Grundlage und erweitern diese mit domänenspezifischen Wissen. Diese Aspekte führen zu neuen Anforderungen bzgl. der Synchronisation von Ontologieänderungen. Auch müssen Abhängigkeiten in der Nutzung von Ontologien beachtet werden. So nutzen beispielsweise in den Lebenswissenschaften viele Analysewerkzeuge oder Applikationen die weit verbreitete Gene Ontology wobei das GO-Konsortium keinen Überblick über die eigentlichen Nutzer hat. Die Unterstützung von Ontologieversionen ist dabei ein bewährtes Mittel, um Stabilität in Anwendungen zu erzeugen. Typischerweise basieren Anwendungen auf einer konkreten Version und können bei Veröffentlichung neuerer Versionen bei Bedarf umgestellt werden. Jedoch ändert sich das Wissen gerade in den Lebenswissenschaften rapide, was zu häufigen Veröffentlichungen neuerer Ontologieversionen führt. Beispielsweise veröffentlicht das Konsortium der GO jeden Tag eine neue Version der Gene Ontology.

Nachfolgend werden Ansätze und Systeme zur Ontologieevolution präsentiert. Überblicksartikel zum Thema Ontologieevolution und Versionierung sind [34, 44, 123].

Das **Protégé** System [87] unterstützt verschiedene Arten der kollaborativen Ontologieentwicklung. So können erstens Ontologien synchron oder asynchron verändert werden. Eine synchrone Entwicklung erfordert es die Ontologie zentral zu verwalten und konkurrierende Änderungen der teilnehmenden Entwickler zu unterstützen. Im Falle einer asynchronen Bearbeitung laden sich Entwickler den Stand der Ontologie und können offline ihre Änderungen umsetzen. Anschließend werden die umgesetzten Änderungen der Nutzer zusammengefasst. Dabei müssen durch parallele Änderungen hervorgerufene Konflikte behoben werden. Zweitens kann eine explizite Versionierung einer Ontologie unterstützt werden oder nicht. Eine Ontologie kann

periodisch archiviert (versioniert) werden. Dies ermöglicht im Fall von Fehlentwicklungen eine Umkehr auf eine ältere, korrekte Ontologieversion. Alternativ werden die Änderungen kontinuierlich direkt in die Ontologie übertragen, d. h. die Ontologie spiegelt jederzeit den aktuellen Stand der Entwicklung wider. Drittens besteht die Möglichkeit Änderungen einer zusätzlichen Qualitätskontrolle (Abnahme) durch Kuratoren oder Domänenexperten zu unterziehen. So können potentielle Probleme gelöst werden, um eine hohe Qualität der Ontologie zu gewährleisten. Typischerweise wird eine solche Kontrolle vor der Veröffentlichung einer neuen Version durchgeführt. Schließlich können Änderungen geloggt werden (Monitoring) oder nicht.

Das hinter dem System befindliche Evolution-Framework unterstützt ein Menge einfacher wie komplexer Änderungsoperationen [87]. Mögliche Änderungen sind in der *Change and Annotation Ontology* (CHAO) klassifiziert. Annotationen beinhalten u. a. den Typ der Änderung, das Konzept, Attribut bzw. die Instanz welche verändert wurden, sowie Nutzer und Datum/Uhrzeit der Änderung. Es existieren zwei Ansätze zur Spezifikation von Änderungen. Einerseits können Änderungen auf Basis eines Logs spezifiziert werden. Andererseits wird lediglich die neue Ontologieversion vorgegeben, d. h. ein Log von Änderungen zwischen der alten und neuen Version liegt nicht vor. Im letzteren Fall wird semi-automatisch ein Diff (Evolution-Mapping mit einer Menge von Änderungen) zwischen den entsprechenden Versionen bestimmt.

Protégé nutzt den PROMPTDIFF Algorithmus [90] um das Evolution-Mapping zwischen zwei Ontologieversionen zu berechnen. Dabei werden die beiden Versionen $V1$ und $V2$ mit Hilfe eines Fixpunktalgorithmus miteinander verglichen. Der Algorithmus wendet schrittweise und wiederholend verschiedene heuristische Matcher (z. B. „same name/type“, „unmatched sibling“ oder „unmatched inverse slots“) an und stoppt falls keine weiteren Änderungen gefunden werden. Die gefundenen Änderungen werden in einer sogenannten „difference table“, welche Elemente der Version $V1$ mit Elementen von $V2$ verknüpft, erfasst. Jedes Tupel der Tabelle stellt dabei eine Änderungsoperation (z. B. add, delete, split, merge oder map) inklusive Parameter dar.

Die verschiedenen Werkzeuge zur Unterstützung der Ontologieevolution stehen in Protégé über zwei Plugins Nutzern zur Verfügung [86, 89]. Das *Change Management Plugin* gestattet Nutzern einen Zugang zu durchgeführten Änderungen, erlaubt die Erfassung von Annotationen zu Änderungen und kann zur Untersuchung der Änderungshistorie eines Konzepts verwendet werden. Über das *PROMPT Plugin* steht der PROMPTDIFF Algorithmus zum Vergleich zweier Ontologieversionen zur Verfügung. Zusätzlich können Nutzer wie Kuratoren oder Domänenexperten vorhandene Änderungen innerhalb einer Prüfung annehmen oder ablehnen. Neben den beiden Plugins verfügt der Protégé Editor über weitere Funktionalitäten wie das Erstellen von Ontologien im Client-Server Modus und biete ein Transaktionsmanagement sowie Undo-Unterstützung.

Der **KAON** Prototyp (Karlsruhe Ontology and Semantic Web Tool Suite) [35, 119]

stellt ein graphisches Nutzerinterface zur inkrementellen Bearbeitung von Ontologien bereit. Der Änderungsprozess umfasst sechs Phasen. Für jede Änderung werden die folgenden Phasen sequentiell durchlaufen: (1) Change Capturing, (2) Change Representation, (3) Semantics of Change, (4) Change Implementation, (5) Change Propagation und (6) Change Validation [113, 114]. Der Evolutionsprozess kann zyklisch sein, d. h. nach der letzten Phase ist ein erneuter Durchlauf für weitere Änderungen möglich.

In der ersten Phase (Change Capturing) entscheidet sich der Entwickler eine bestimmte Änderung an der Ontologie durchzuführen, z. B. das Löschen eines Konzepts. In Phase zwei (Change Representation) wird die Änderungsanforderung aus Phase eins in eine formalere Repräsentationsform überführt. Dabei unterscheidet der Ansatz zwischen elementaren (einfachen) und zusammengesetzten (komplexen) Änderungen. Letztere können durch eine Zusammensetzung mehrerer elementarer Änderungen ausgedrückt werden. Insgesamt wird zwischen 16 elementaren Änderungen (Löschung / Einfügung / Modifikation von Konzepten, Eigenschaften, Axiomen und Beziehungen) und 12 zusammengesetzten Änderungen (Zusammenfassen und Reklassifikation von Konzepten, Extraktion / Kopieren von Konzepten, ...) unterschieden.

Phase drei verwendet die formale Repräsentation der Änderung, um mögliche Konflikte (Inkonsistenzen), welche durch die Ausführung der Änderung in der Ontologie entstehen, zu identifizieren. So hat die Löschung eines Konzepts c einen Einfluss auf dessen Subkonzepte und Instanzen. Zur Auflösung solcher Inkonsistenzen können verschiedene Evolutionsstrategien (Optionen) definiert werden. Beispielsweise können die Subkonzepte von c auch gelöscht werden oder sie können unter dem Vaterkonzept von c eingeordnet werden. Um den manuellen Aufwand für diese Art von Entscheidungen zu reduzieren, können Standard-Evolutionsstrategien spezifiziert werden. Diese werden dann beim Eintreten einer bestimmten Inkonsistenz ausgeführt. Des Weiteren können Evolutionsstrategien auf Basis von generellen Designzielen, z. B. Minimierung der Anzahl von Änderungen oder Vermeidung von tiefen Ontologien, auch automatisch vom System generiert werden.

Die Menge durchzuführender Änderungen aus Phase zwei und drei wird anschließend dem Entwickler zur Bestätigung präsentiert. Bei positiver Bestätigung werden die Änderungen in Phase vier umgesetzt (implementiert). Alle Änderungen werden in einem Versionslog protokolliert, eine explizite Versionierung findet nicht statt. Die nachfolgende Phase fünf (Propagation) ist für die Propagierung der Änderungen in abhängige Applikationen oder Ontologien verantwortlich. Der Ansatz nimmt an, dass alle Konsumenten der Ontologie bekannt sind, d. h. der Evolutionsprozess kann in abhängigen Ontologien rekursiv angewandt werden. Die finale Validierungsphase gibt Entwicklern die Möglichkeit die implementierten Änderungen zu überprüfen und mittels Undo ungewollte Änderungen zurückzunehmen. Nach Abschluss dieser Phase können weitere Änderungen mittels Start eines neuen Evolutionsprozesses umgesetzt werden.

Das **OntoView** System [62, 63] fokussiert auf die Versionierung RDF-basierter Ontologien. Die Versionierung lehnt sich an das in der Softwareentwicklung häufig eingesetzte Concurrent Versioning System (CVS)[6] an. Einer der Kernkomponenten ist der strukturelle Vergleich von Ontologieversionen zur Bestimmung von Änderungen (Berechnung eines Evolution-Mapping). Dabei werden die folgenden Arten von Änderungen unterschieden. „Non-logical changes" betreffen Modifikationen am Label oder an Kommentaren eines Konzepts. „Logical definition changes" ändern die Semantik eines Konzepts, z. B. Änderungen in subClassOf Beziehungen, der Domain/Range von Properties oder Einschränkungen von Properties. Sonstige Änderungen umfassen Modifikationen an Identifiern sowie das Einfügen/Löschen von Definitionen. Komplexere Änderungen wie das Zusammenfassen oder Splitten von Konzepten werden nicht unterstützt.

Der Erkennungsalgorithmus ähnelt der in UNIX verfügbaren diff Funktion, jedoch wird in OntoView eine Graphstruktur in Form von RDF Tripeln (<subject, predicate, object>) als Basis für den Vergleich der Versionen verwendet. Mit Hilfe sogenannter „IF-THEN" Regeln, welche vordefinierte Bedingungen auf der alten und neuen Version auswerten, werden Änderungen zwischen den Versionen detektiert. Sind sowohl die Bedingungen auf der alten wie auch neuen Version erfüllt, wird eine entsprechende Änderung im Diff Ergebnis registriert. Die Autoren behaupten, dass auf Basis dieser Technik nahezu alle Arten von Änderungen (mit Ausnahme von Änderungen an Identifiern) erkannt werden können.

Der Ansatz in [100, 101] baut auf dem Evolutionsprozess von KAON auf. Der vorgestellte Evolutionsprozess besteht aus fünf Phasen: (1) Change Request, (2) Change Implementation, (3) Change Detection, (4) Change Recovery und (5) Change Propagation. Der Hauptunterschied zu [114] liegt in der Change Detection Phase wo zusätzlich implizite Änderungen auf Basis eines Logs vorheriger Änderungen erkannt werden. Zudem enthält der Versionslog die verschiedenen Versionen eines Konzepts, d. h. Informationen über die Änderungshistorie eines Konzepts.

Änderungen sind entweder einfach oder zusammengesetzt (komplex) und werden deklarativ mit Hilfe der *Change Definition Language* (CDL) spezifiziert. Einfache wie zusammengesetzte Änderungen werden auf Basis von Informationen aus der alten und neuen Ontologieversion sowie durch die regel-basierten Änderungsdefinitionen erkannt. So beschreibt z. B. folgende Definition

$$\forall p \in P, a \in C : addDomain(p, A) \leftarrow \neg hasDomain(p, A, v_{i-1}) \wedge hasDomain(p, A, v_i)$$

die einfache Änderung *addDomain(p,A)*, welche einer Property p die Domain A zuweist. Eine solche Änderung liegt genau dann vor, wenn die Domain in Version v_{i-1} noch nicht vorhanden war jedoch in Version v_i. Zusammengesetzte (komplexe) Änderungen sind weitaus schwieriger zu erkennen, da sie in der Regel mehrere Ontologieelemente umfassen. Zudem ist die Erkennung komplexer Änderungen ein

[6] CVS: `http://www.nongnu.org/cvs`

wichtiger Punkt, um u. a. Instanzen korrekt zu migrieren. Beispielsweise würden zwei einfache Änderungen eine Property p aus Konzept $c1$ in das Konzept $c2$ verschieben und anschließend würde eine Beziehung (subClassOf) zwischen $c1$ und $c2$ eingefügt werden. Falls man obige Änderungen als einzeln (separat) ansieht, so müssten nach Schritt eins alle p Properties in Instanzen von $c1$ entfernt werden. Jedoch würde die nachfolgende Einfügung der subClassOf Beziehung eine Ergänzung von p Properties in Instanzen von $c1$ erfordern. Findet man heraus, dass die beiden einfachen Änderungen zu einer zusammengesetzten Änderung *moveUpProperty* von p gehören, hätten unnötige Löschungen von p Einträgen vermieden werden können.

Die Bestimmung von High-Level Änderungen in RDF/S Ontologien wurde in [98] untersucht. Das vorgeschlagene Framework nutzt eine formale Änderungssprache und unterscheidet dabei zwischen einfachen, zusammengesetzten und heuristischen Änderungen. Heuristische Änderungen (z. B. Merge, Split oder Umbenennungen von Konzepten) werden durch die Anwendung heuristischer Matcher bestimmt. Der Algorithmus zur Änderungserkennung fokussiert auf einfache und zusammengesetzte Änderungen und setzt auf einem Low-Level Delta (Diff) zwischen zwei Versionen $v1$ und $v2$ auf. Dieses Delta ist die symmetrische Differenz zwischen $v1$ und $v2$ und enthält somit alle eingefügten bzw. gelöschten RDF-Tripel. Änderungen werden durch drei Eigenschaften definiert: (1) Menge eingefügter RDF-Tripel, (2) Menge gelöschter RDF-Tripel und (3) eine Menge von Bedingungen, welche erfüllt sein müssen. Beispielsweise wird die Änderung *Delete_SuperClass(x,y)*, welche eine Subklassenbeziehung zwischen zwei Konzepten x und y entfernt wie folgt beschrieben: (1) keine eingefügten RDF-Tripel existieren, (2) ein gelöschtes Tripel (x, subClassOf y) liegt vor und (3) x ist ein Konzept in $v1$. Der Diff-Algorithmus bestimmt zunächst potentielle Änderungen zwischen $v1$ und $v2$ auf Basis der vorliegenden Änderungsdefinitionen sowie dem Low-Level Delta. Im zweiten Schritt werden iterativ Änderungen, welche die Bedingungen erfüllen selektiert und das Low-Level Delta entsprechend reduziert. Dabei werden zunächst zusammengesetzte und später einfache Änderungen behandelt, d. h. es liegt eine Priorisierung (Ausführungsreihenfolge) bei der Änderungserkennung vor.

Im Bereich der Lebenswissenschaften existieren spezifische Werkzeuge wie OBO-Edit [25] oder OBO to OWL [82], welche die Entwicklung von OBO Ontologien ermöglichen. OBO-Edit ist ein im GO Konsortium entwickelter Editor zur graphischen Anzeige und Entwicklung biomedizinischer Ontologien im OBO Format. Änderungen umfassen u. a. das Einfügen neuer Konzepte und Beziehungen bzw. das Modifizieren von Attributen wie Name, Synonyme oder Definition. Durchgeführte Änderungen werden geloggt und ermöglichen Entwicklern somit ein Undo von Änderungen innerhalb einer Session. OBO to OWL ist ein für Protégé entwickeltes Plugin und ermöglicht über eine Schnittstelle die Entwicklung von OBO Ontologien in Protégé. Das Plugin besteht aus einem Mapping Programm, einem Interface zur Protégé OWL API sowie einem graphischen Nutzerinterface. Das Mapping Programm implementiert einen OBO Parser, welcher Ontologien im OBO Format

zunächst in ein generisches Ontologieformat und später in äquivalente Konstrukte in OWL überführt. Nach dem Editieren der Ontologie in Protégé kann diese wieder ins OBO Format zurückgeschrieben und exportiert werden. Die längerfristige Evolution von Ontologien wurde mit Hilfe einfacher Statistiken am Beispiel der GO untersucht [122]. Es fand jedoch keine Änderungserkennung zwischen veröffentlichten Ontologieversionen statt.

2.5 Zusammenfassung und Abgrenzung der eigenen Arbeit

Die Unterstützung einer effektiven Schemaevolution ist ein langwieriges Problem, da Schemaänderungen typischerweise weitreichende Folgen besitzen, z. B. Einflüsse auf Instanzen, Indexe, Speicherstrukturen wie auch Anwendungen und sonstige auf einem Schema basierende Komponenten. Dieses Kapitel führte zunächst in die Thematik der Schemaevolution ein und gab anschließend einen Überblick über Arbeiten zur Schemaevolution für relationale Schema, XML sowie Ontologien.

Kommerzielle Datenbanksysteme unterstützen Schemaevolution an relationalen Schemas derzeit lediglich mit einfachen inkrementellen Änderungsoperationen und einer zugehörigen Anpassung von Instanzen. Es gibt beispielsweise keine Unterstützung für die semi-automatische Propagierung von Änderungen in abhängige Schemas, Mappings oder Applikationen. Um diese Lücke zu schließen, muss eine Bestimmung und Anwendung ausdrucksstarker Mappings wie beispielsweise in Forschungsprototypen (Pantha Rei/PRISM, ...) oder dem Model Management praktiziert, erfolgen.

Die Evolution von XML Schemas gestaltet sich in der Regel leichter als die von relationalen Schemas, da die Schemas häufig durch optionale Elemente erweitert werden können und in diesen Fällen keine Migration von XML Dokumenten (Instanzen) nötig ist. Da für XML Schemas noch keine standardisierte Sprache für Änderungen vorliegt, wird häufig lediglich das neue Schema direkt angegeben. In einigen Forschungsansätzen werden Techniken des Schema-Matching und -Mapping zur semi-automatischen Bestimmung eines Evolution-Mapping zwischen zwei Schemaversionen angewandt. Zusätzlich kann daraus ein ausführbares Mapping zur Migration der Instanzen abgeleitet werden. Ähnlich wie bei relationalen Schemas gibt es wenig Unterstützung für die Propagierung von Änderungen in abhängige Schemas oder Mappings.

Bei der Ontologieevolution wird sowohl die inkrementelle Änderung als auch die direkte Bereitstellung neuer Ontologieversionen verfolgt. Im letzteren Fall wurden bereits einige Werkzeuge zur semi-automatischen Bestimmung des Evolution-Mappings zwischen einer alten und neuen Ontologieversion vorgeschlagen. Die resultierenden Mappings beinhalten ein Menge von einfachen und auch teilweise komplexen Än-

derungen. Während die Anpassung von Instanzen, falls diese mit der Ontologie verwaltet werden, schon untersucht wurde, haben Ansätze zur Propagierung von Änderungen in andere abhängige Ontologien bzw. Anwendungen noch recht wenig Aufmerksamkeit erhalten.

Gerade in den Lebenswissenschaften ist aufgrund der zahlreichen Ontologien, deren Größe/Komplexität und des enorm schnellen Wissenszuwachses eine effektive Unterstützung von Ontologieevolution erforderlich. Nutzer müssen häufig mit den Konsequenzen neu veröffentlichter Versionen einer Ontologie ohne Unterstützung umgehen können. Aus diesen Gründen geht diese Dissertation insbesondere auf die in den Lebenswissenschaften vorhandenen Probleme der Ontologieevolution ein. Tab. 2.1 zeigt dazu einen Vergleich der vorliegenden Arbeit mit existierenden Systemen basierend auf den in Kapitel 1 dargestellten Anforderungen. Der Fokus dieser Arbeit liegt einerseits auf der quantitativen Evolutionsanalyse von Ontologien wie auch Ontologie-basierter Mappings. Um sich ein Bild über langfristige Änderungen und Trends machen zu können, schlägt diese Arbeit Methoden zur quantitative Evolutionsanalyse für verschiedene Zeiträume vor (siehe Kapitel 4 und 7). Die anderen dargestellten Systeme fokussieren auf Probleme wie die kollaborative Ontologieentwicklung (Protégé), die Unterstützung des Evolutionsprozesses (KAON) oder das Verwalten von Versionen (OntoView). Die Durchführung quantitativer Evolutionsanalysen erfordert aufgrund der enormen Größe der Ontologien sowie der hohen Anzahl an Versionen eine effiziente Versionierung, welche in Kapitel 8 vorgestellt wird. Es findet eine sequentielle Versionierung bereits existierender Ontologieversionen statt, wobei verschiedene Ontologieformate unterstützt werden. Im Vergleich erlauben die beiden Systeme Protégé und OntoView ebenfalls eine sequentielle Versionierung von Ontologien.

In dieser Arbeit geht es primär nicht darum Ontologien weiterzuentwickeln (anzupassen), wie beispielsweise in Protégé oder KAON praktiziert, sondern die Konsequenzen der Evolution zu betrachten. Hierzu werden zwei Algorithmen zur Differenzbestimmung zwischen zwei Ontologieversionen sowie zur Lokalisierung (in)stabiler Teile innerhalb einer Ontologie vorgeschlagen. Die in Kapitel 5 beschriebene regelbasierte Änderungsbestimmung auf Basis eines Match zwischen zwei Ontologieversionen ist in der Lage einfache wie auch komplexe Änderungen für ein Evolution-Mapping zu bestimmen. Im Gegensatz dazu unterstützt OntoView nur die Erkennung einfacher Änderungen. In KAON können zwar komplexe Änderungen umgesetzt werden, allerdings gibt es kein Werkzeug zur Differenzbestimmung zwischen zwei Ontologieversionen. Der in Protégé vorhandene PROMPTDIFF Algorithmus ähnelt dem in dieser Arbeit vorgeschlagenen Verfahren, jedoch besitzt der regelbasierte Ansatz eine enorme Flexibilität, d. h. es können flexibel neue Änderungsoperationen definiert und mit Hilfe entsprechender Regeln erkannt werden. Dies ist insbesondere für Ontologien in den Lebenswissenschaften wertvoll, da hier oftmals spezielle Änderungen wie z. B. Modifikationen am Zustand eines Konzepts (aktiv oder obsolete) vorliegen und entsprechend erkannt werden müssen. In keinem der anderen Systeme

	Protégé	KAON	OntoView	Vorliegende Arbeit *
Beschreibung / Fokus	Flexibles Werkzeug zur Ontologieentwicklung und -evolution	Unterstützung des Evolutionsprozesses für eine konsistente Evolution von Ontologien	Versionsmanagement und -vergleich für RDF Ontologien	Quantitative Analyse der Evolution von Ontologien und Mappings in den Lebenswissenschaften (Kapitel 4)
Unterstützte Ontologieformate	RDF/OWL, weitere Formate über Import Plugins	RDF/OWL	RDF	OBO, RDF, CSV; weitere Formate durch adaptierbaren Import
Änderungsoperationen *1) Umfang (einfach, komplex)* *2) Spezifikation (inkrementell, neue Version)*	1) Einfach und komplex (moveClass, addSuperClass ...) 2) Inkrementell oder Spezifikation einer geänderten Ontologieversion	1) Einfach und komplex (merge, copy, ...) 2) Inkrementell	1) Einfach 2) Integration neuer Versionen	1) Einfach und komplex (merge, split, ...) 2) Integration neuer Versionen
Evolution-Mapping *1) Darstellung* *2) Differenzbestimmung*	1) Liste inkrementeller Änderungen oder Tabelle mit Differenzen zwischen zwei Versionen 2) PROMPTDIFF	1) Liste inkrementeller Änderungen 2) -	1) Menge von Änderungen zur Beschreibung der Differenz zweier Versionen 2) Regelbasierte Änderungserkennung	1) Menge von Änderungen zur Beschreibung der Differenz zweier Versionen; Unterscheidung zwischen einfachem und komplexem Evolution-Mapping 2) Regelbasierte Differenzbestimmung auf Basis eines Match beider Ontologieversionen (Kapitel 5) sowie Unterstützung zur Bestimmung änderungsintensiver Ontologieregionen (Kapitel 6)
Propagierung von Änderungen *1) Instanzen / Mappings* *2) Abhängige Ontologien*	1) - 2) -	1) Migration von Instanzen welche mit der Ontologie verwaltet werden 2) Rekursive Anwendung des Evolutionsprozesses auf abhängigen Ontologien	1) - 2) -	1) Migration veralteter Annotation-Mappings (Kapitel 7) 2) -
Versionierung	Sequentielle Versionierung	-	Sequentielle Versionierung auf der Basis von CVS	Sequentielle Versionierung existierender Ontologien (Kapitel 8)
Infrastruktur / GUI	Protégé Editor mit PROMPT and Change Management Plugin	GUI / Editor in der KAON Infrastruktur	Webbasierte Applikation um Ontologieversionen zu versionieren und zu vergleichen	Webanwendung OnEX zur Exploration von Änderungen in Ontologien der Lebenswissenschaften (Kapitel 7)

Tabelle 2.1: Vergleich der eigenen Arbeit mit bestehenden Systemen (* vorliegende Arbeit bezieht sich auf die Ergebnisse der Kapitel 4–8)

wurde bisher die Bestimmung änderungsintensiver bzw. stabiler Ontologieregionen thematisiert. Das in Kapitel 6 vorgestellte neuartige Verfahren ermöglicht eine solche Bestimmung, was insbesondere bei großen Ontologien wie in den Lebenswissenschaften eine kompakte und verständlichere Erfassung der Evolution für Entwickler und Anwender gestattet.

Die Propagierung von Ontologieänderungen in abhängige Daten wie Instanzen oder Mappings wird teilweise in KAON angeboten. In dieser Arbeit wird insbesondere die Migration von Annotation-Mappings, welche auf einer alten Ontologieversion basieren, unterstützt (siehe Kapitel 7). Vergleichbar mit den anderen Systemen bietet die webbasierte Applikation OnEX (siehe Kapitel 7) ein graphisches Nutzerinterface für die Gesamtinfrastruktur an. Nutzer können u. a. die Evolution von Ontologien in den Lebenswissensschaften analysieren und nachvollziehen sowie die Änderungshistorie von Konzepten untersuchen.

3 Grundlagen und Modelle

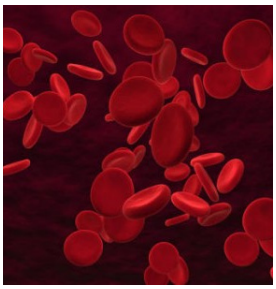
In diesem Kapitel erfolgt die Definition und Beschreibung der grundlegenden Modelle, welche im restlichen Teil der Arbeit Verwendung finden. Zunächst werden das Ontologiemodell sowie die Versionierung von Ontologien eingeführt. Des Weiteren werden Instanz- und Mappingmodelle inklusive ihrer verschiedenen Arten erläutert.

3.1 Ontologiemodell

3.1.1 Ontologie

Eine *Ontologie* O kann wie folgt beschrieben werden: $O = (C, A, R)$. Dabei werden Konzepte aus C, welche Attribute aus A besitzen, durch gerichtete Beziehungen aus R miteinander verbunden. Die Ontologie stellt einen gerichteten azyklischen Graphen (directed acyclic graph – DAG) dar, wobei ein Konzept mehrere Vorgängerkonzepte besitzen kann (Mehrfachvererbung). Spezielle Wurzelkonzepte ($roots \in C$) haben keine Beziehung zu einem Vorgängerkonzept und repräsentieren somit die obersten Konzepte einer Ontologie. Es wird angenommen, dass eine Ontologie exakt ein Wurzelkonzept besitzt. Ist dies nicht der Fall, wird ein sogenanntes virtuelles Wurzelkonzept ($virtual\_root$) eingeführt und alle existierenden Wurzelkonzepte werden als dessen Kinder definiert.

Jedes Konzept $c \in C$ einer Ontologie wird durch ein Menge von Attributen aus A definiert und näher beschrieben. Ein Attribut $a = (a_{concept}, a_{name}, a_{value})$ eines Konzepts $a_{concept}$ besitzt einen Namen a_{name} und einen entsprechenden Attributwert



blood coagulation

accession:	GO:0007596
name:	blood coagulation
synonym(s):	blood clotting
obsolete:	false
definition:	The sequential process by which the multiple coagulation factors of the blood interact, ultimately resulting in the formation of an insoluble fibrin clot; it may be divided into three stages: stage 1, the formation of intrinsic and extrinsic prothrombin converting principle; stage 2, the formation of thrombin; stage 3, the formation of stable fibrin polymers.

Abbildung 3.1: Konzept „blood coagulation“ aus GO-BP mit zugehörigen Attributen

a_{value}. Typische Attribute sind der Name (Label) eines Konzepts, dessen Definition sowie Synonyme. Ein spezielles ID-Attribut (id) wird verwendet, um ein Konzept innerhalb der Ontologie wie auch übergreifend eindeutig zu identifizieren. Dieses Attribut kann vorgegeben sein, beispielsweise wird in Ontologien der Lebenswissenschaften eine *accession* Nummer zur eindeutigen Kennzeichnung von Konzepten verwendet. Falls dieses Attribut nicht vorliegt, muss es erzeugt werden.

R beinhaltet gerichtete Beziehungen $r = (r_{source}, r_{type}, r_{target})$, dabei werden die beiden Konzepte r_{source} und r_{target} über den semantischen Typ r_{type} miteinander verbunden. Der am häufigsten anzutreffende semantische Typ ist is_a. is_a Beziehungen verkörpern eine Spezialisierung, d. h. spezifischere Konzepte werden über is_a Beziehungen mit allgemeineren Konzepten verbunden. Sie bilden somit die strukturelle Basis einer jeden Ontologie. Darüber hinaus werden weitere Beziehungen wie z. B. $part\_of$ oder has_parts verwendet, um domänenspezifische Sachverhalte zu modellieren (z. B. Teil-Ganzes Beziehungen im Bereich der Anatomien).

Auf Basis der zuvor eingeführten Notation, kann beispielsweise die Subontologie Biologische Prozesse der Gene Ontology wie folgt beschrieben werden: $GO_{BP} = (C, A, R)$. Die Wurzel ($root$) bildet das Konzept GO:0008150 (biological_process). Abb. 3.1 zeigt beispielhaft das Konzept „blood coagulation“, welches den biologischen Prozess Blutgerinnung widerspiegelt. Neben dem Namen besitzt das Konzept die eindeutige ID (accession) GO:0007596 sowie ein Synonym „blood clotting“. Das Konzept wird als aktiv eingestuft (obsolete=false), d. h. es ist aktuell und kann in Analysen oder Anwendungen verwendet werden. Eine ausführlichere Definition des Konzepts in natürlicher Sprache erläutert dessen Bedeutung im Detail.

Die strukturelle Einordnung des Konzepts GO:0007596 innerhalb der Ontologie wird über die Menge der Beziehungen R von GO_{BP} festgelegt. Die mit Hilfe von Ami-

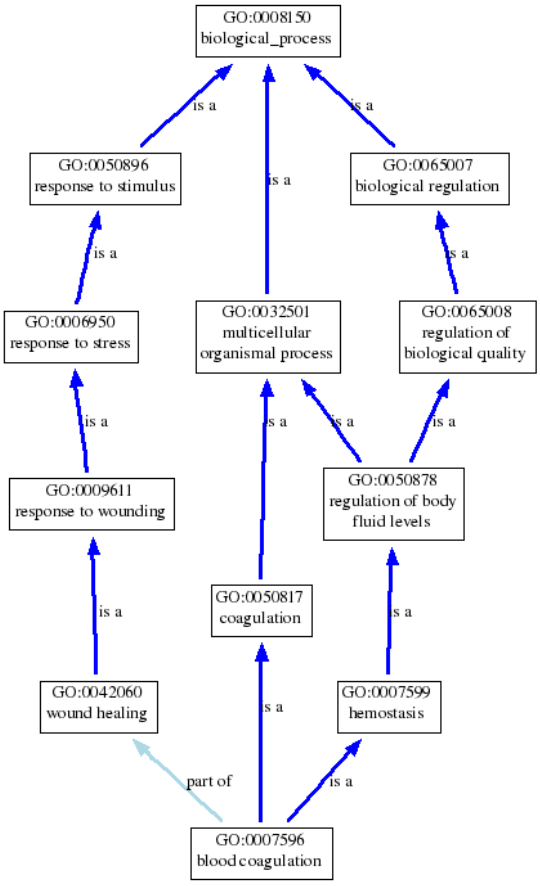


Abbildung 3.2: Strukturelle Einordnung des Konzepts GO:0007596 in GO_{BP} (generiert mit AmiGO)

GO[7] erzeugte Abb. 3.2 zeigt die strukturelle Einordnung von GO:0007596 innerhalb von GO_{BP}. Das Konzept besitzt drei Vorgängerkonzepte: is_a Beziehungen zu GO:0050817 (coagulation) und GO:0007599 (hemostasis) sowie eine $part\_of$ Verbindung zu GO:0042060 (wound healing). Durch Mehrfachvererbung entstehen vier verschiedene Pfade von GO:0007596 zur Wurzel der Ontologie.

Zusammengefasst weist Konzept GO:0007596 die folgenden Attribute und Beziehungen (nur Vorgänger dargestellt) in der eingeführten Notation auf:

- (GO:0007596, *accession*, GO:0007596)
- (GO:0007596, *name*, blood coagulation)
- (GO:0007596, *synonym*, blood clutting)
- (GO:0007596, *obsolete*, false)

[7] AmiGO: `http://amigo.geneontology.org`

- (GO:0007596, *definition*, The sequential process by which ...)
- (GO:0007596, *is_a*, GO:0050817)
- (GO:0007596, *is_a*, GO:0007599)
- (GO:0007596, *part_of* ,GO:0042060)

3.1.2 Ontologieversionen

Eine *Ontologieversion* $O_v = (C_v, A_v, R_v, t)$ in der Version v repräsentiert den Zustand (Inhalt) der Ontologie zum Zeitpunkt der Veröffentlichung t. Die Elemente (Konzepte C_v, Attribute A_v und Beziehungen R_v) von O_v sind solange gültig bis eine neuere Version v' mit $t' > t$ freigegeben wird. Dies geschieht meist in regelmäßigen Abständen (z. B. monatlich am Monatsende) oder wann immer aufgrund der durchgeführten Änderungen eine Veröffentlichung notwendig erscheint. Beispielsweise ist für GO täglich eine neue Version abrufbar[8], wohingegen das National Cancer Instiute seinen Thesaurus auf einer monatlichen Basis[9] veröffentlicht.

Es wird angenommen, dass die Versionen einer Ontologie einem linearem Versionierungsschema folgen. So besitzt jede Ontologieversion O_i exakt eine Vorgänger- (O_{i-1}) sowie eine Nachfolgeversion (O_{i+1}). Nur die erste (initiale) Ontologieversion besitzt keine Vorgängerversion und die aktuelle (letzte) Version hat keine Nachfolgeversion. Bei einer monatlichen Versionierung hat z. B. die Juli 2010 Version der Biologischen Prozesse in GO ($GO_{BP,2010-07}$) die Juni Version ($GO_{BP,2010-06}$) als Vorgänger und die August Version ($GO_{BP,2010-08}$) als Nachfolger.

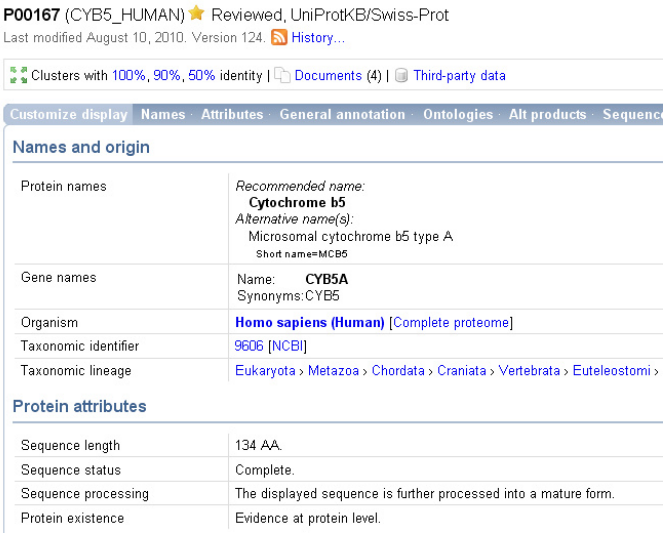
Die zuvor zum Konzept GO:0007596 aus GO_{BP} gezeigten Informationen (Attribute und Beziehungen) waren u.a. in der Version $GO_{BP,2010-07}$ gültig. Hingegen wies das Konzept in früheren Versionen zum Teil veränderte Informationen auf. So wurde die is_a Beziehung zu GO:0050817 erstmalig in der Version $GO_{BP,2004-01}$ eingeführt und blieb seither unverändert. Im Gegensatz dazu existierte im Zeitraum zwischen den Versionen $GO_{BP,2005-09}$ und $GO_{BP,2007-11}$ ein weiteres Synonym namens „blood coagulation factor activity“.

3.2 Instanzmodell

Neben Ontologien existieren *Instanzquellen* $IS = (I)$, welche eine Menge von Instanzen I, z. B. Proteine oder Gene in den Lebenswissenschaften beinhalten. Analog zu Konzepten einer Ontologie besitzen Instanzen $i \in I$ eine ID zur eindeutigen Identifikation innerhalb der Instanzquelle. Die ID wird z. B. in Mappings zur Erfassung von

[8] `http://archive.geneontology.org/latest-termdb/`
[9] `http://evs.nci.nih.gov/ftp1/NCI_Thesaurus/archive`

Korrespondenzen verwendet (siehe nachfolgender Abschnitt). Des Weiteren werden quellenspezifische Attribute zur Beschreibung der Instanzen genutzt. Beispielsweise besitzt das Protein P00167 (Cytochrome b5) in SwissProt[10], neben seiner ID, Informationen über den bevorzugten Namen, Alternativnamen, Sequenzlänge oder den Status der Sequenz (siehe Abb. 3.3).



P00167 (CYB5_HUMAN) Reviewed, UniProtKB/Swiss-Prot
Last modified August 10, 2010. Version 124. History...

Clusters with 100%, 90%, 50% identity | Documents (4) | Third-party data

Customize display | Names | Attributes | General annotation | Ontologies | Alt products | Sequenc

Names and origin

Protein names	*Recommended name:* **Cytochrome b5** *Alternative name(s):* Microsomal cytochrome b5 type A Short name=MCB5
Gene names	Name: **CYB5A** Synonyms: CYB5
Organism	**Homo sapiens (Human)** [Complete proteome]
Taxonomic identifier	9606 [NCBI]
Taxonomic lineage	Eukaryota › Metazoa › Chordata › Craniata › Vertebrata › Euteleostomi ›

Protein attributes

Sequence length	134 AA.
Sequence status	Complete.
Sequence processing	The displayed sequence is further processed into a mature form.
Protein existence	Evidence at protein level.

Abbildung 3.3: Informationen zum Protein P00167 (Cytochrome b5) in SwissProt

Auch für Instanzquellen werden regelmäßige Versionen veröffentlicht. Die Version einer Instanzquelle $IS_v = (I_v, t)$ beinhaltet alle zum Zeitpunkt t gültigen Instanzen I_v. Entsprechend der Versionierung von Ontologien wird eine lineare Versionierung angenommen, d. h. jede Version besitzt exakt eine Vorgänger- bzw. Nachfolgeversion.

3.3 Mappingmodell

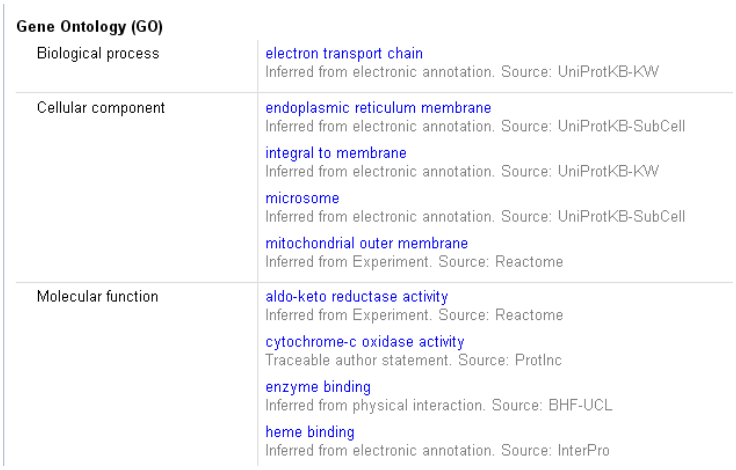
Unter einem *Mapping* wird i. Allg. eine Menge von Korrespondenzen zwischen den Objekten zweier Datenquellen verstanden. Basierend auf dem Typ der Datenquelle (Ontologie oder Instanzquelle) können verschiedene Mappingarten unterschieden werden. Die verschiedenen Mappingarten werden in den folgenden Abschnitten definiert und genauer diskutiert.

3.3.1 Annotation-Mappings

Ein Mapping zwischen einer Instanzquelle und einer Ontologie wird als *Annotation-Mapping* AM bezeichnet. Ein Annotation-Mapping $AM = (IS_u, O_v, Corr)$ verbin-

[10] `http://www.uniprot.org/uniprot/P00167`

det die Instanzquelle IS in Version u mit der Ontologieversion O_v. Die Korrespondenzen (oder Annotationen) $Corr$ in einem Annotation-Mapping setzen jeweils eine Instanz i aus IS_u mit einem Konzept c aus O_v in Beziehung $(i, c) \in Corr$, d. h. das Instanzobjekt i wird mit c annotiert. Annotationen werden üblicherweise zur Klassifikation bzw. zur semantisch eindeutigen Beschreibung von Instanzobjekten genutzt. In den Lebenswissenschaften werden u.a. Konzepte der Gene Ontology zur semantisch einheitlichen Beschreibung von Proteinen verwendet. Als ein Beispiel zeigt Abb. 3.4 die GO Annotationen für das Protein P00167 (Cytochrome b5) in der Datenquelle SwissProt[11]. So ist beispielsweise das Protein am Biologischen Prozess „electron transport chain“ beteiligt und agiert an den Zellulären Komponenten „microsome“ oder „integral to membrane“. Auch in anderen Domänen wie dem e-Buisness oder dem Bibliothekswesen werden Konzepte aus Ontologien zur Annotation entsprechender Instanzobjekte (z. B. Produkte, Bücher) eingesetzt. So werden angebotene Produkte in eBay mit Kategorien wie „Drives & Storage“ oder „PC Laptops & Netbooks“ verknüpft, um ein einfachere Suche und Navigation nach Produkten zu ermöglichen.



Gene Ontology (GO)

Biological process	electron transport chain Inferred from electronic annotation. Source: UniProtKB-KW
Cellular component	endoplasmic reticulum membrane Inferred from electronic annotation. Source: UniProtKB-SubCell
	integral to membrane Inferred from electronic annotation. Source: UniProtKB-KW
	microsome Inferred from electronic annotation. Source: UniProtKB-SubCell
	mitochondrial outer membrane Inferred from Experiment. Source: Reactome
Molecular function	aldo-keto reductase activity Inferred from Experiment. Source: Reactome
	cytochrome-c oxidase activity Traceable author statement. Source: ProtInc
	enzyme binding Inferred from physical interaction. Source: BHF-UCL
	heme binding Inferred from electronic annotation. Source: InterPro

Abbildung 3.4: GO Annotationen des Proteins P00167 (Cytochrome b5) in SwissProt

3.3.2 Ontologie-Mappings

Liegt ein Mapping zwischen zwei Ontologien vor, so spricht man von einem *Ontologie-Mapping*. Ein Ontologie-Mapping $OM = (X_u, Y_v, Corr, Method)$ verbindet zwei Ontologieversionen X_u und Y_v über eine Menge von Korrespondenzen $Corr$, welche mittels einer konkreten Methode $Method$ bestimmt wurden. Dabei wird ein Konzept x_k aus X_u mit einem Konzept y_k aus Y_v durch eine Korrespondenz $(x_k, y_k, sim_k) \in Corr$ verbunden. Der Ähnlichkeitswert sim_k wurde mittels

[11] `http://www.uniprot.org/uniprot/P00167#section_terms`

der spezifizierten *Method* bestimmt und gibt an wie stark die beiden Konzepte zusammenhängen (d. h. wie ähnlich sie sind). Der Wert kann zwischen 0 und 1 liegen, dabei bedeutet 0 keine Ähnlichkeit wohingegen ein Wert von 1 exakte Gleichheit bedeutet. Für die Bestimmung von Ontologie-Mappings werden üblicherweise semi-automatische Match-Verfahren verwendet (siehe [30, 104] für einen Überblick). Beispielsweise verwenden metadaten-basierte Match-Algorithmen [107] ausschließlich Informationen aus den Ontologien (z. B. Konzeptnamen oder Strukturinformationen) zur Bestimmung von Korrespondenzen zwischen Konzepten. Andere Verfahren wie instanz-basierte Matcher [61] beziehen Informationen aus Instanzen ein, z. B. die Anzahl zugeordneter Instanzen zu einem Konzept, um die Ähnlichkeit zwischen Konzepten zu berechnen. Die Methode kann auch manuell sein, falls beispielsweise durch einen Experten ein Mapping erstellt bzw. validiert wurde.

Ontologie-Mappings sind eine wichtige Ressource zur Integration heterogener Datenbestände [71, 58] und werden in Analysen / Studien der Lebenswissenschaften zur Beantwortung komplexer Forschungsfragen verwendet. Ein Beispiel ist das Ontologie-Mapping zwischen der Adult Mouse Anatomy [54] und dem Anatomieteil des NCI Thesaurus [109]. Dieses Mapping ermöglicht Wissenschaftlern vergleichende Studien, u. a. inwieweit sich Krankheiten in der Maus als Modelle für Krankheiten im Menschen einsetzen lassen. Die Bestimmung dieses Mappings ist Bestandteil der jährlichen Ontology Alignment Evaluation Initiative (OAEI)[12], welche versucht die Stärken und Schwächen existierender Match-Systeme zu analysieren, um verbesserte Match-Verfahren zu entwerfen. Die erzielten Ergebnisse der Systeme werden jährlich publiziert (für 2009 siehe [108]) und stehen öffentlich Wissenschaftlern zur Verfügung[13,14,15].

[12] `http://oaei.ontologymatching.org/`
[13] Anatomy Track 2010: `http://webrum.uni-mannheim.de/math/lski/anatomy10/`
[14] Anatomy Track 2009: `http://webrum.uni-mannheim.de/math/lski/anatomy09/`
[15] Anatomy Track 2008: `http://webrum.uni-mannheim.de/math/lski/anatomy08/`

Teil II

Algorithmen zur Änderungsbestimmung

4

Vergleichende Evolutionsanalyse von Ontologien und Mappings

4.1 Motivation

Mit der steigenden Bedeutung und Verwendung von Ontologien, z. B. zur Annotation von Objekten oder für Datenintegrationsaufgaben (bzw. Analysen via Ontologie-Mappings), geht eine stetige Evolution der Ontologien einher. Aufgrund neuer Erkenntnisse aus Forschung und veränderter Anforderungen müssen Ontologien entsprechend angepasst werden, um stets einen aktuellen und konsistenten Wissenstand der jeweiligen Domäne zu repräsentieren. Typische Änderungsoperationen umfassen dabei das Einfügen neuer Konzepte und Beziehungen, aber auch die Löschung veralteter Konzepte und Beziehungen. Um Nutzern von Ontologien ein uneingeschränktes Weiterarbeiten in den Anwendungen zu gewährleisten, werden Versionen von Ontologien veröffentlicht, d. h. ein Nutzer kann weiterhin auf einer alten Version arbeiten und seine Anwendungen erst zu einem späteren Zeitpunkt auf eine neuere Version umstellen. Somit stellt jede Ontologieversion den Stand (Status) der Ontologie zum Zeitpunkt ihrer Veröffentlichung dar. Während ältere Ontologieversionen eventuell einen stabilen Charakter, d. h. wenig Änderungen aufweisen, kann eine neue Version durchaus eine große Menge wichtiger Änderungen umfassen, z. B. durch eine Umstellung eines Teils der Ontologiestruktur. Diese Änderungen können dann eine direkte Auswirkung auf existierende Anwendungen wie Analysen besitzen, welche beispielsweise Ontologie-basierte Annotation- oder Ontologie-Mappings nutzen. Somit ist eine Identifizierung von Änderungen in Ontologien und abhän-

gigen Annotation- und Ontologie-Mappings nötig, um entsprechende Korrekturen bzw. Anpassungen einleiten zu können. Des Weiteren sollte neu hinzugekommenes Wissen in den Ontologien so schnell wie möglich in Analysen usw. ausgenutzt werden.

Die Untersuchung der Evolution von Ontologien in den Lebenswissenschaften sowie deren Einfluss auf Annotation- bzw. Ontologie-Mappings wurde bisher kaum untersucht (siehe verwandte Arbeiten in Kapitel 2). Dieses Kapitel beschäftigt sich zunächst mit einer grundlegenden quantitativen Evolutionsanalyse existierender Ontologien und Mappings in den Lebenswissenschaften. So können u. a. die folgenden Fragen beantwortet werden: „Wie (in)stabil sind verschiedene Ontologien?“, „Wie häufig treten die diversen Änderungsarten während der Evolution auf?“ oder „Welche strukturellen Änderungen finden typischerweise in Ontologien statt?“ . Des Weiteren sollen die Konsequenzen von Änderungen an Ontologien untersucht werden, z. B. inwieweit diese in Änderungen an Annotation- und Ontologie-Mappings resultieren bzw. deren Stabilität beeinflussen.

Das vorliegende Kapitel umfasst die folgenden Beiträge:

- Es wird ein generisches Framework zur systematischen Evolutionsanalyse von Ontologien und Instanzquellen (z. B. Protein- oder Gendatenbanken) sowie Mappings (Annotation- und Ontologie-Mappings) vorgeschlagen. Das Framework umfasst Metriken, welche zur Beschreibung und Einschätzung der Evolution von Ontologieversionen und Mappings eingesetzt werden.
- In einer umfassenden Analyse wurde das Framework für eine vergleichende Untersuchung der Evolution in 16 Ontologien der Lebenswissenschaften eingesetzt. Die vorgeschlagenen Metriken wurden zur Bestimmung wichtiger Änderungen und anderer Aspekte der Evolution angewandt.
- Ebenfalls wurde die Evolution von Annotation-Mappings sowie die Korrelation zwischen Änderungen in den Ontologien/Instanzen und den Ontologie-basierten Annotationen untersucht. Eine Evolutionsanalyse für automatisch erzeugte Ontologie-Mappings rundet die Untersuchungen ab.

4.2 Evolutionsmodell und Metriken des Frameworks

In diesem Abschnitt werden das Evolutionsmodell sowie die Metriken des Frameworks beschrieben. Für Evolutionsanalysen unterscheidet das Framework zwischen zwei Typen. Einerseits werden einzelne Datenquellen bzgl. ihrer Evolution untersucht. Hierzu gehören Ontologien und Instanzquellen, welche in Versionen vorliegen. Auf der anderen Seite wird die Analyse der Evolution von Mappings ermöglicht.

Dies betrifft Annotation-Mappings, d. h. Assoziationen zwischen Instanzquellen und Ontologien, sowie Ontologie-Mappings, d. h. Assoziationen zwischen verschiedenen Ontologien. Für die Modelle wird auf das Grundlagenkapitel (Kapitel 3) verwiesen. Im nachfolgenden wird zunächst das Evolutionsmodell erläutert und danach die Metriken des Frameworks definiert.

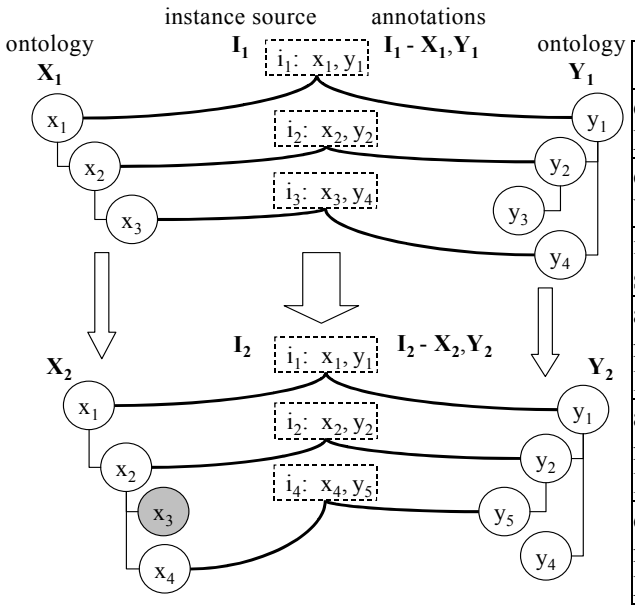
4.2.1 Evolutionsmodell

Die Analyse der Evolution in einzelnen Datenquellen und Mappings erfordert ein generisches Evolutionsmodell, welches auf alle Modelle (Instanzquellen, Ontologien, Annotation-Mappings, Ontologie-Mappings) anwendbar ist. Die Basis des Modells bilden Mengen von Elementen E_{vi} einer Version v_i. Mögliche Elemente die Evolution unterliegen, umfassen Konzepte und Beziehungen (Ontologien), Instanzdaten (Instanzquellen), Annotationen (Annotation-Mappings) und Korrespondenzen (Ontologie-Mappings).

Es werden grundsätzlich drei mögliche Änderungsoperationen unterschieden: *add*, *del* und *toObs*. Während *add* das Einfügen neuer Elemente in eine Datenquelle oder ein Mapping beschreibt, findet mittels *del* das Löschen veralteter oder nicht mehr benötigter Elemente statt. Die Änderungsoperation *toObs* ist eine in Ontologien angewandte Operation, um Konzepte als veraltet zu kennzeichnen. Im Gegensatz zu *del* werden dabei veraltete Konzepte nicht physisch gelöscht, sondern verbleiben in der Datenquelle. Nutzer der Ontologie sollten die als veraltet markierten Konzepte nicht mehr aktiv verwenden. Beispielsweise sollte eine Ontologie-basierte Analyse nicht solche Konzepte einschließen. Zur Einfachheit und mit dem Ziel der gleichzeitigen Analyse der Evolution in einzelnen Datenquellen und Mappings werden keine komplexeren Änderungsoperationen im Framework einbezogen. Die Erkennung komplexer Änderungen wie beispielsweise das Verschieben eines Konzepts (move) innerhalb einer Ontologie wird in einem gesonderten Kapitel (Kapitel 5) besprochen.

Um die Evolution quantitativ untersuchen zu können, wird für jede der drei Änderungsoperationen die Menge betroffener Elemente in den entsprechenden Versionen bestimmt:

- $add_{vi,vj} = E_{vj} \backslash E_{vi}$: eingefügte Elemente zwischen Version v_i und v_j
- $del_{vi,vj} = E_{vi} \backslash E_{vj}$: gelöschte Elemente zwischen Version v_i und v_j
- $toObs_{vi,vj} = E_{vj,obs} \cap E_{vi,nonObs}$: Elemente, welche zwischen v_i und v_j als veraltet markiert wurden. In diesem Fall werden mit den Mengen $E_{vi,obs}$ bzw. $E_{vi,nonObs}$ die aktiven (normalen) und veralteten Elemente innerhalb der Version v_i unterschieden. Zusammen bilden die beiden Mengen die Menge aller Elemente in der Version v_i: $E_{vi} = E_{vi,obs} \cup E_{vi,nonObs}$.



source	*add*	*del*	*toObs*
ontology X	{x_4}	{}	{x_3}
ontology Y	{y_5}	{y_3}	{}
instance source I	{i_4}	{i_3}	{}
annotation mapping I-X	{i_4-x_4}	{i_3-x_3}	{}
annotation mapping I-Y	{i_4-y_5}	{i_3-y_4}	{}
ontology mapping X-Y	{x_4-y_5}	{x_3-y_4}	{}

Abbildung 4.1: Beispiel einer Evolution mit Ontologien (X,Y), Instanzquelle (I), Annotation-Mappings (I-X,Y) und einem Ontologie-Mapping (X-Y)

Diese Elementmengen können für Ontologien, Instanzquellen sowie Mappings wie folgt bestimmt werden. Die verfügbaren IDs (accessions) von Elementen werden zum Abgleich der Elemente der beiden Versionen verwendet. Wenn ein Element mit einer bestimmten ID nur in der neuen Version vorkommt, wird dieses der *add*-Menge zugeordnet. Im Gegensatz dazu werden Elemente, die nur in der alten Versionen vorhanden sind, in der *del*-Menge erfasst.

Ein Beispiel für Ontologie-, Instanz- sowie Mapping-Evolution ist in Abb. 4.1 dargestellt. Das Beispiel zeigt die Evolution zweier Ontologien X (von X_1 nach X_2) und Y (von Y_1 nach Y_2), einer Instanzquelle I (von I_1 nach I_2), zweier Annotation-Mappings $I-X$ (von I_1-X_1 nach I_2-X_2) und $I-Y$ (von I_1-Y_1 nach I_2-Y_2) und ein Ontologie-Mapping $X-Y$ (von X_1-Y_1 nach X_2-Y_2). So existiert in X_2 ein neues Konzept x_4, wohingegen x_3 als veraltet markiert wurde. Für x_4 wurde ebenfalls eine neue Annotation (i_4-x_4) eingefügt und eine neue Korrespondenz (x_4-y_5) ist verfügbar. Für x_3 wurden die zugehörige Annotation (i_3-x_3) und Korrespondenz (x_3-y_4) aus den Mappings entfernt.

4.2.2 Metriken des Frameworks

Auf Basis des eingeführten Evolutionsmodells werden nun eine Reihe von Metriken zur Analyse der Datenquellen (Ontologien, Instanzquellen) und Mappings, deren Evolution und ihres Wachstums definiert. Es werden zunächst die Metriken zur sta-

tistischen Auswertung von Datenquellen und Mappings eingeführt. Danach erfolgt eine Erläuterung der Evolutions- und Wachstumsstatistiken.

Deskriptive Statistiken für Datenquellen und Mappings

Für alle Elementmengen (Instanzen, Konzepte, Beziehungen, Annotationen, Korrespondenzen) kann deren Kardinalität für eine bestimmte Version einer Instanzquelle, Ontologie oder eines Mappings bestimmt werden. Für Ontologien werden zusätzlich strukturelle Aspekte wie die verwendeten Beziehungstypen (z. B. is_a und part_of), Konzepttypen (veraltet vs. aktiv, Blatt vs. inneres Konzept), Eingangs- bzw. Ausgangsgrade sowie die Anzahl und Länge von Pfaden über weitere Metriken erfasst. Die folgenden Metriken stehen zur Verfügung:

- $|E_{vi}|$: Anzahl von *Elementen* E in einer Version v_i einer Datenquelle oder eines Mappings $E \in \{$Konzepte C, Beziehungen R, Instanzdaten I, Annotation- bzw. Ontologie-Mapping $A\}$
- $|C_{leaf}|, |C_{inner}|$: Anzahl von Blattkonzepten bzw. inneren Konzepten
- $|C_{obs}|, |C_{nonObs}|$: Anzahl aktiver bzw. veralteter Konzepte
- $|R_{is\_a}|, |R_{part\_of}|, |R_{mis}|$: Anzahl von is_a, part_of oder sonstigen Beziehungen
- $\varnothing d_{in} = \frac{|C_{inner}|}{|R_{is\_a}|+|R_{part\_of}|}$: durchschnittlicher Eingangsgrad von inneren Konzepten
- $\varnothing d_{out} = \frac{|C|}{|R_{is\_a}|+|R_{part\_of}|}$: durchschnittlicher Ausgangsgrad von Konzepten
- $\varnothing ppc, \varnothing ppl$: durchschnittliche Anzahl von Pfaden pro Konzept oder Blattkonzept (unter einem Pfad wird der Weg von einem Konzept zur Wurzel der Ontologie entlang von is_a und part_of Beziehungen verstanden)
- $\varnothing pl, \varnothing pl_{leaf}$: durchschnittliche Pfadlänge aller Konzepte bzw. aller Blattkonzepte

Für ein Mapping A seien die folgenden Elementmengen $X_{A,u} \subseteq X_u$ und $Y_{A,v} \subseteq Y_v$ in zwei Versionen u und v gegeben. Dabei verbindet A jedes Element aus $X_{A,u}$ mit mindestens einem Element aus $Y_{A,v}$ und umgekehrt besitzt jedes Element aus $Y_{A,v}$ ein Gegenstück in $X_{A,u}$. Auf Basis dessen kann die relative Abdeckung („coverage“) von X_u und Y_v bzgl. des Mappings A bestimmt werden, d. h. der Anteil an Elementen in X_u bzw. Y_v, die durch mindestens eine Korrespondenz in A abgedeckt werden. Die Metriken cov_{A,X_u} und cov_{A,Y_v} erlauben die Bestimmung der relativen Abdeckung der Elemente in X_u bzw. Y_v durch A:

- $cov_{A,X_u} = \frac{|X_{A,u}|}{|X_u|}$
- $cov_{A,Y_v} = \frac{|Y_{A,v}|}{|Y_v|}$

Metriken zur Analyse von Evolution und Wachstum

Die Metriken zur Analyse der Evolution setzen auf dem generischen Evolutionsmodell (siehe Abschnitt 4.2.1) auf. Somit sind diese auf allen eingeführten Modellen (Ontologien, Instanzquellen, Mappings) anwendbar. Um die Anzahl von Änderungen zu erfassen, können einerseits zwei Versionen v_i und v_j direkt miteinander verglichen werden, d. h. im Ergebnis wird die Anzahl von Änderungen betroffener Elemente erfasst. Alternativ kann eine Analyse der Änderungen auch für ein vorgegebenes Zeitintervall erfolgen, beispielsweise alle Änderungen innerhalb einer Periode p oder bzgl. eines regulären Zeitintervalls t innerhalb von p (z. B. pro Monat oder pro Jahr).

- $Add_{v_i,v_j} = |add_{v_i,v_j}|$: Anzahl eingefügter Elemente zwischen Version v_i und v_j
- $Del_{v_i,v_j} = |del_{v_i,v_j}|$: Anzahl gelöschter Elemente zwischen Version v_i und v_j
- $Obs_{v_i,v_j} = |toObs_{v_i,v_j}|$: Anzahl von Elementen, welche zwischen Version v_i und v_j auf veraltet geändert wurden
- $Add_{p,t}$,$Del_{p,t}$,$Obs_{p,t}$: durchschnittliche Anzahl *eingefügter / gelöschter / veraltet gesetzter* Elemente pro Zeitintervall t innerhalb der Zeitperiode p

Auf Basis der obigen Metriken kann der relative Anteil („fraction“) eingefügter bzw. gelöschter Elemente sowie ein add-del Quotient (adr) für zwei Versionen bestimmt werden. Des Weiteren können die relativen Anteile bzgl. eines Zeitintervalls t innerhalb einer Zeitperiode p berechnet werden:

- $adr_{v_i,v_j} = \frac{Add_{v_i,v_j}}{Del_{v_i,v_j}+Obs_{v_i,v_j}}$: add-del Quotient für Änderungen zwischen v_i und v_j
- $add\text{-}frac_{v_i,v_j} = \frac{Add_{v_i,v_j}}{|E_{v_j}|}$: Anteil der Elemente in Version v_j, welche zwischen v_i und v_j eingefügt wurden
- $del\text{-}frac_{v_j,v_i} = \frac{Del_{v_i,v_j}}{|E_{v_i}|}$: Anteil der Elemente in Version v_i, welche zwischen v_i und v_j gelöscht wurden
- $obs\text{-}frac_{v_i,v_j} = \frac{Obs_{v_i,v_j}}{|E_{v_j}|}$: Anteil der Elemente in Version v_j, welche zwischen v_i und v_j auf veraltet geändert wurden

- *add-frac*$_{p,t}$, *del-frac*$_{p,t}$, *obs-frac*$_{p,t}$: durchschnittliche Anteile *eingefügter / gelöschter / veraltet gesetzter* Elemente pro Zeitintervall t innerhalb der Zeitperiode p auf Basis der versionsbasierten $frac$-Metriken

Des Weiteren werden Metriken zur Erfassung des Wachstums („growth“) einer Quelle oder eines Mappings definiert:

- $growth_{E,v_i,v_j} = \frac{|E_{v_j}|}{|E_{v_i}|} \in [0, \infty] \subseteq \Re$, $|E_{v_i}| > 0$

Das Wachstum wird dabei durch den Quotienten der Anzahl der Elemente in v_j zu denen in v_i bestimmt. Die Elemente können wiederum alle zuvor beschriebenen Typen annehmen: Konzepte C, Beziehungen R, Instanzdaten I, Annotationen A und Ontologie-Mapping A. Das Wachstum weist einen Wert >1 auf, falls ein Anstieg in der Anzahl der Elemente vorliegt. Ein Rückgang von Elementen resultiert in einem Wachstum <1. Falls keine Änderungen zwischen zwei Versionen v_i und v_j vorliegen, ist das Wachstum 1: $growth_{v_i,v_j} = 1$. Das Wachstum kann zudem auch auf den relativen Metriken wie beispielsweise der Abdeckung angewandt werden. So würde ein Anstieg der Abdeckung einer Ontologie bzgl. eines Mappings von 50% auf 60% ein Wachstum von 1,2 bedeuten.

4.3 Evolutionsanalyse von Ontologien

Die Evolutionsanalyse greift verschiedene Ontologien aus unterschiedlichen Domänen der Lebenswissenschaften auf. So werden die beiden bekannten und weit verbreiteten Ontologien Gene Ontology (GO) [36] und NCI Thesaurus [109] aber auch spezifischere Ontologien aus der OBO Foundry [110] wie Sequence Ontology[16] oder Zebrafish Anatomy [111] bzgl. ihrer Evolution untersucht. Für die Analyse wurde ein zentrales Repository zur Verwaltung der Ontologien und ihrer Versionen eingesetzt. Details zur Versionierung der Ontologien innerhalb des Repositories können im Kapitel 8 eingesehen werden. Im Rahmen der Analyse wurden insgesamt 386 Versionen von 16 Ontologien aus den Lebenswissenschaften integriert.

Nachfolgend wird zunächst ein Überblick über die analysierten Ontologien und deren Versionen präsentiert. Anschließend werden die in Abschnitt 4.2.2 definierten Metriken zur Analyse der Evolution angewandt. Exemplarische Verläufe (Trendcharts), welche die Evolution der beiden Subontologien Molekulare Funktionen und Biologische Prozesse der GO zeigen, werden in Abschnitt 4.4 diskutiert. Trendcharts und detaillierte Statistiken für alle in der Analyse untersuchten Ontologien können online[17] oder im Anhang A eingesehen werden. Zudem besteht die Möglichkeit mit Hilfe des OnEX Systems (siehe Kapitel 7) aktuelle Trendcharts zu generieren.

[16] `http://www.sequenceontology.org/`
[17] `http://dbs.uni-leipzig.de/ls_ontology_evolution`

Ontology	size	$\|C\|_{start}$	$\|C\|_{last}$	$grow_{\|C\|, start, last}$	t_{start}	t_{last}	k	characteristics, domain and use
NCI Thesaurus	large	35,814	63,924	1.78	May. 04	Dec. 07	39	broad coverage of cancer domain
GeneOntology		17,368	25,995	1.50	May. 04	Feb. 08	44	aggregation of all GO sub ontologies
-- Biological Process		8,625	15,001	1.74	May. 04	Feb. 08	44	annotation of gene products (biological role)
-- Molecular Function		7,336	8,818	1.20	May. 04	Feb. 08	44	annotation of gene products (molecular function)
-- Cellular Components		1,407	2,176	1.55	May. 04	Feb. 08	44	annotation of gene products (cellular location)
ChemicalEntities		10,236	18,007	1.76	Oct. 04	Jan. 08	28	chemical compounds of biological relevance
FlyAnatomy	medium	6,090	6,222	1.02	Nov. 04	Dec. 07	16	anatomy of Drosophila melanogaster
MammalianPhenotype		4,175	6,077	1.46	Aug. 05	Jan. 08	15	terms for annotating mammalian phenotypic data
AdultMouseAnatomy		2,416	2,745	1.14	Aug. 05	Sep. 07	15	adult anatomy of the mouse (Mus)
ZebrafishAnatomy		1,389	2,172	1.56	Nov. 05	Oct. 07	12	anatomy and development of the Zebrafish
Sequence		981	1,463	1.49	Aug. 05	Feb. 08	26	structured CV for sequence annotation
ProteinModification		1,074	1,128	1.05	Jun. 06	Nov. 07	14	description of protein chemical modifications
CellType	small	687	857	1.25	Jun. 04	Jun. 07	19	cell types from prokaryotes to mammals
PlantStructure		681	835	1.23	Jul. 05	Feb. 08	22	plant morphological and anatomical structures
ProteinProteinInteraction		194	819	4.22	Aug. 05	Feb. 08	19	annotation of protein interaction experiments
FlyBaseCV		658	693	1.05	Nov. 05	Apr. 07	7	used for various aspects of annotation by FlyBase
Pathway		427	593	1.39	Nov. 05	Jan. 08	22	CV for pathways, annotation of gene products
Overall		***82,190***	***131,530***	***1.60***			***386***	

Tabelle 4.1: Überblick und Statistiken zur Versionierung der untersuchten Ontologien

4.3.1 Überblick

Tab. 4.1 zeigt alle untersuchten Ontologien inklusive Details über deren Größe, Anzahl Versionen innerhalb der Untersuchungsperiode, das Wachstum sowie Informationen über die Domäne. Aus Übersichtsgründen wurden die Ontologien auf Basis ihrer Größe $|C|$ in drei Gruppen eingeteilt: *large* ($|C|$>10.000), *medium* (1.000<$|C|$<10.000) und *small* ($|C|$<1.000). Die gesamte Analyse umfasst einen Zeitraum von 45 Monaten zwischen Mai 2004 und Februar 2008. Die beiden Zeitstempel t_{start} und t_{last} der ersten bzw. letzten Version sowie die Anzahl von Versionen (k) geben Auskunft über die Häufigkeit der Versionierung einer Ontologie, d. h. wie oft werden neue Versionen veröffentlicht und wie lange sind diese gültig. Während einige Ontologien wie GO täglich eine neue Version herausgeben, wird in der Analyse maximal eine Version pro Monat in Betracht gezogen (im Falle mehrerer Versionen wird die zuerst veröffentlichte Version verwendet). Die beiden großen Ontologien GO und NCI Thesaurus sind ebenfalls die ältesten Ontologien innerhalb der Analyse und weisen die meisten Versionen auf. Hingegen wurden andere Ontologien wie FlyBaseCV oder CellType seit geraumer Zeit nicht mehr aktualisiert (ca. 6–8 Monate). Dies zeigt, dass diese Ontologien scheinbar einen nahezu finalen Zustand erreicht haben. Im Schnitt besitzt jede Ontologie im Analysezeitraum 25 Versionen, d. h. eine Version ist üblicherweise nicht länger als 2 Monate gültig.

Mit Blick auf die Anzahl der Konzepte in den Ontologien ist ein starkes Wachstum innerhalb des Analysezeitraums zu beobachten. Im Durchschnitt nahm die Anzahl der Konzepte in den letzten 45 Monaten um 60% zu, wobei das stärkste (geringste) Wachstum 4,22 (1,02) betrug. Die derzeit größte Ontologie innerhalb der Analyse

	Full period (May. 04 - Feb. 08)							Last year (Feb. 07 - Feb. 08)		
Ontology	*Add*	*Del*	*Obs*	*adr*	*add-frac*	*del-frac*	*obs-frac*	*Add*	*Del*	*Obs*
NCI Thesaurus	627	2	12	42.4	1.3%	0.0%	0.0%	416	0	5
GeneOntology	200	12	4	12.2	0.9%	0.1%	0.0%	222	20	5
-- Biological Process	146	7	2	16.2	1.2%	0.1%	0.0%	133	10	2
-- Molecular Function	36	3	2	6.8	0.4%	0.0%	0.0%	69	7	3
-- Cellular Components	18	2	0	8.9	1.0%	0.1%	0.0%	19	3	0
ChemicalEntities	256	62	0	4.1	1.8%	0.5%	0.0%	384	67	0
FlyAnatomy	5	1	1	3.3	0.1%	0.0%	0.0%	6	0	0
MammalianPhenotype	65	2	9	6.0	1.2%	0.0%	0.2%	74	2	3
AdultMouseAnatomy	11	0	0	30.9	0.4%	0.0%	0.0%	1	0	0
ZebrafishAnatomy	33	5	1	5.5	1.8%	0.3%	0.1%	45	2	1
Sequence	19	3	2	4.1	1.5%	0.3%	0.2%	19	0	0
ProteinModification	5	2	1	1.5	0.4%	0.2%	0.1%	7	0	2
CellType	5	1	0	2.8	0.7%	0.2%	0.1%	1	0	0
PlantStructure	5	0	1	6.1	0.7%	0.0%	0.1%	3	0	0
ProteinProteinInteraction	21	0	0	41.7	2.7%	0.0%	0.2%	4	0	0
FlyBaseCV	1	0	1	2.1	0.2%	0.0%	0.1%	0	0	0
Pathway	7	1	0	7.9	1.3%	0.2%	0.0%	6	2	0

Tabelle 4.2: Evolution untersuchter Ontologien (Intervall t=1 Monat)

NCI Thesaurus wuchs um 80% auf nahezu 64.000 Konzepte an. Innerhalb der GO sind die Biologischen Prozesse (BP) die größte und am stärksten wachsende Subontologie (74% Wachstum). Hingegen sind die Molekularen Funktionen (MF) lediglich um 20% innerhalb des Beobachtungszeitraumes gewachsen.

Tab. 4.2 zeigt detaillierte und zeit-normalisierte Statistiken der Evolution der Ontologien auf. Dabei werden die durchschnittliche Anzahl eingefügter, gelöschter und veralteter Konzepte pro Monat für den kompletten Analysezeitraum sowie für das letzte Jahr (Feb. 2007–Feb. 2008) dargestellt. Zusätzlich sind die relativen Anteile von eingefügten, gelöschten und veralteten Konzepten pro Monat angegeben.

Die größten Ontologien (Gruppe *large*) weisen ebenfalls die größten Änderungszahlen auf. Im Durchschnitt werden in diesen Ontologien 360 (25) Konzepte pro Monat eingefügt (gelöscht) im Gegensatz zu 86 (6) Einfügungen (Löschungen) in allen Ontologien. Des Weiteren ist zu erkennen, dass die Einfügung von Konzepten die dominierende Änderungsoperation in allen untersuchten Ontologien ist. Jedoch weisen einige Ontologien, z. B. ChemicalEntities oder GO, ebenfalls eine signifikante Anzahl von Löschungen auf. Der Quotient zwischen Einfügungen und Löschungen(add-del-Quotient *adr*) ist ein Indikator für die Häufigkeit des Auftretens der beiden Operationen. NCI Thesaurus besitzt mit 42 den größten *adr*-Wert aller Ontologien, d. h. es finden im Schnitt 42-mal mehr Einfügungen als Löschungen / Veraltet-Markierungen statt. Im Gegensatz dazu fällt der Quotient mit 4 für ChemicalEntities am geringsten aus, d. h. 20% aller Änderungen sind Löschungen. Die relativen Metriken $add\text{-}frac$, $del\text{-}frac$ und $obs\text{-}frac$ zeigen, dass einige mittelgroße oder kleine Ontologien durch-

aus große Änderungsraten besitzen. Beispielsweise werden in der ProteinProtein Interaction Ontology monatlich 2,7% neue Konzepte eingefügt.

Eine andere interessante Beobachtung betrifft die Verwendung des „Obsolete"-Mechanismus in den unterschiedlichen Ontologien. Einige der untersuchten Ontologien setzen den Mechanismus nicht ein, d. h. Konzepte werden sofort aus der Ontologie entfernt, z. B. in ChemicalEntities oder AdultMouse Ontology. Die meisten Ontologien sind flexibel, d. h. Konzepte können je nach Bedarf gelöscht oder lediglich als veraltet markiert werden. Insbesondere NCI Thesaurus und MammalianPhenotype nutzen den „Obsolete"-Mechanismus, anstatt veraltete oder nicht mehr benötigte Konzepte direkt zu löschen.

Der Vergleich der Ergebnisse für den gesamten Analysezeitraum mit den Ergebnissen für das letzte Jahr ermöglicht eine Abschätzung von Trends in der Evolution der einzelnen Ontologien. Eine erste Gruppe von Ontologien weist in beiden Zeiträumen hohe Änderungsfrequenzen auf z. B. NCI Thesaurus, GO oder MammalianPhenotype. Das Domänenwissen dieser Ontologien ändert sich ständig und entwickelt sich weiter, d. h. die Ontologien umfassen Wissen sehr aktiver Forschungsgebiete. Diese werden voraussichtlich in naher Zukunft durch neue Erkenntnisse und Ergebnisse die Entwicklung der Ontologien weiterhin vorantreiben. Andere Ontologien besitzen im letzten Jahr verglichen mit dem gesamten Analysezeitraum eine erhöhte Änderungsfrequenz z. B. ChemicalEntities oder GO-MF. Dies deutet auf größere Umgestaltungen der Ontologien oder ansteigende Forschungsaktivitäten in den entsprechenden Gebieten hin. Ein letzte Gruppe umfasst Ontologien, welche im vergangenen Jahr nur wenig oder gar nicht geändert wurden z. B. AdultMouse Ontology, CellType und FlyBaseCV. Die Entwicklung dieser Ontologien kann somit als nahezu abgeschlossen angesehen werden, d. h. die aktuellen Versionen sind als fast stabil einzuschätzen.

4.3.2 Einfluss der Evolution auf die Ontologiestruktur

In diesem Abschnitt werden Änderungen in der Struktur analysiert und diskutiert. Tab. 4.3 fasst dazu alle relevanten Metriken für die erste und letzte Version der größten Ontologien zusammen. Des Weiteren werden Wachstumsraten im unteren Drittel der Tabelle präsentiert. Für die Untersuchung der strukturellen Änderungen werden die folgenden Eigenschaften in Betracht gezogen: Anteil innerer vs. Blattkonzepte, die Anzahl und der Anteil von is_a, part_of und anderen Beziehungen sowie die Eingangs-/Ausgangsgrade von Konzepten und die Anzahl von Pfaden.

Die Untersuchung zeigt, dass der Großteil von Konzepten Blätter sind, d. h. Konzepte, welche keine weiteren Spezialisierungen innerhalb der Ontologie besitzen. Jedoch hat sich der Anteil von Blattkonzepten im Analysezeitraum von 70% auf 67% reduziert. Gleichermaßen bedeutet dies einen Zuwachs innerer Konzepte und damit eine Zunahme der Komplexität der Ontologiestrukturen. In der letzten GO-BP Version liegen aufgrund einer starken Abnahme ($growth_{|C_{leaf}|}$=0,88) sogar weniger Blatt-

	Ontology	$\|C_{leaf}\|$ (%)	$\|R\|$	$\|R_{isa}\|$ (%)	$\|R_{partof}\|$ (%)	$\|R_{mis}\|$ (%)	$\varnothing\, d_{out}$	$\varnothing\, d_{in}$	$\varnothing\, pl_{leaf}$	$\varnothing\, ppl$
First version	*NCI Thesaurus*	79	41,281	100			1.2	5.6	8.2	3.3
	GeneOntology	66	23,589	88	12		1.4	4.0	7.3	3.7
	– Biological Process	52	13,358	85	15		1.5	3.2	8.0	7.1
	– Molecular Function	82	8,459	100			1.2	6.4	5.3	1.4
	– Cellular Components	67	1,772	52	48		1.3	3.8	5.5	1.8
	ChemicalEntities	70	11,593	100			1.1	3.8	8.3	2.3
	MammalianPhenotype	68	4,620	100			1.1	3.4	5.5	1.5
Last version	*NCI Thesaurus*	79	72,466	100			1.1	5.4	8.0	3.0
	GeneOntology	60	41,396	88	12		1.6	3.8	8.6	22.9
	– Biological Process	46	27,141	84	16		1.8	3.3	8.8	38.7
	– Molecular Function	81	10,195	100			1.2	5.9	6.2	1.7
	– Cellular Components	64	4,060	79	21		1.9	5.0	8.3	52.6
	ChemicalEntities	69	31,233	76	1	23	1.4	4.3	12	18.6
	MammalianPhenotype	64	6,875	100			1.2	3.1	7.5	2.5
Growth	*NCI Thesaurus*	1.00	1.8	1.00			1.0	1.0	1.0	0.9
	GeneOntology	0.91	1.8	1.00	1.03		1.2	1.0	1.2	6.2
	– Biological Process	0.89	2.0	0.99	1.06		1.2	1.0	1.1	5.5
	– Molecular Function	1.00	1.2	1.00			1.0	0.9	1.2	1.2
	– Cellular Components	0.95	2.3	1.51	0.44		1.5	1.3	1.5	28.7
	ChemicalEntities	0.99	2.7	0.76	undef.	undef.	1.3	1.1	1.4	8.0
	MammalianPhenotype	0.95	1.5	1.00			1.1	0.9	1.4	1.7

Tabelle 4.3: Strukturelle Änderungen in den untersuchten Ontologien

konzepte (46%) als innere Konzepte vor.

Die Anzahl an Beziehungen wächst ähnlich stark oder sogar schneller als die Anzahl von Konzepten (siehe Tab. 4.1). Der größte Zuwachs ist in ChemicalEntities zu beobachten ($growth_{|R|}$=2,7 für Beziehungen vs. $growth_{|C|}$=1,76 für Konzepte). Die größten Ontologien werden von is_a Beziehungen dominiert ($\approx$ 91% aller Beziehungen). Hingegen treten part_of (4%) und sonstige Beziehungen (5%) nur selten auf[18]. Einige der Ontologien sind reine is_a Hierarchien, z. B. NCI Thesaurus, GO-MF oder MammalianPhenotype. Die größten Änderungen in den Beziehungen liegen in ChemicalEntities vor. Die Ontologie startete als eine reine is_a Ontologie und wurde im Analysezeitraum nach und nach mit anderen Beziehungen erweitert. Zwischen den Subontologien der GO sind einige Unterschiede bzgl. der Struktur auszumachen. Während die Molekularen Funktionen lediglich is_a Beziehungen verwenden, werden in den Biologischen Prozessen und Zellulären Komponenten zusätzlich part_of Beziehungen zur Modellierung des Wissens eingesetzt. Diese entwickelten sich jedoch unterschiedlich. In GO-BP ist der Anteil von part_of Beziehungen am Ende des Analysezeitraum größer als zu Beginn ($growth_{R_{part\_of}}$=1,06). In GO-CC fand hingegen eine starke Reduzierung des part_of Anteils statt ($growth_{R_{part\_of}}$=0,44).

Darüber hinaus wurde die Entwicklung der durchschnittlichen Anzahl ein- bzw. aus-

[18] In allen Ontologien liegt die folgende Verteilung der Beziehungen vor: 86%, 7%, 7%.

gehender Beziehungen an Konzepten (Eingangs-/Ausgangsgrade) analysiert. Dabei zeigten insbesondere is_a Ontologien nur leichte Veränderungen. Der Ausgangsgrad von Konzepten ist in diesen Ontologien immer kleiner als 1,2, d. h. die Konzepte besitzen in der Regel nur ein Elternkonzept. Im Gegensatz dazu weisen Konzepte in GO-BP oder GO-CC einen Ausgangsgrad von fast zwei in der letzten Version auf, d. h. im Schnitt besitzt jedes Konzept nahezu zwei Elternkonzepte, da gleichzeitig is_a und part_of zur Modellierung von Wissen verwendet wird.

Eine letzte Analyse beschäftigt sich mit Änderungen in den Pfaden insbesondere deren Anzahl und Länge für Blattkonzepte. Bis auf den NCI Thesaurus hat sich die durchschnittliche Pfadlänge von Blattkonzepten in nahezu allen Ontologien um bis zu 50% vergrößert. Die durchschnittliche Anzahl von Pfaden pro Blattkonzept ($\varnothing ppl$) stieg insbesondere in Ontologien, welche nicht nur is_a verwenden mit einem durchschnittlichen Wachstum von ca. 14 enorm an. Der größte Zuwachs ist mit 28 in den Zellulären Komponenten der GO zu beobachten. Dies ist vermutlich das Ergebnis einer größeren strukturellen Umstellung von GO-CC was bereits zuvor beim Vergleich von is_a zu part_of zu erkennen war. Insgesamt zeigen letztere Analysen, dass die Ontologien mehr und mehr strukturiertes Wissen erfassen (z. B. mehr innere Konzepte, Zunahme an Pfaden, Einfügungen neuer Beziehungstypen) und somit eine sukzessive Verfeinerung stattfindet.

4.4 Evolutionsanalyse von Mappings

In diesem Abschnitt wird die Evolutionsanalyse von Mappings dargestellt. Es wird zunächst ein Überblick über das Analyseszenario gegeben, danach erfolgt eine detaillierte Darstellung der Ergebnisse für Instanzen, Annotation- und Ontologie-Mappings.

4.4.1 Analyseszenario

Abb. 4.2 zeigt einen schematischen Überblick des Analyseszenarios. Das Szenario umfasst zwei Ontologien der GO, die Molekularen Funktionen (GO-MF) und die Biologischen Prozesse (GO-BP). Beide werden zur einheitlichen und konsistenten Beschreibung der Eigenschaften von Proteinen verwendet, d. h. Proteine aus Instanzquellen werden mit Konzepten aus beiden Ontologien assoziiert (annotiert). Im Rahmen der Analyse diente Ensembl [33] als Instanzquelle. Ensembl enthält umfangreiche Informationen zu Proteinen des Menschen (Homo Sapiens), u. a. auch Annotationen von Proteinen zu GO-MF und GO-BP, welche bzgl. Evolution analysiert wurden. Für die Analyse der Evolution von Ontologie-Mappings wurden mit Hilfe von metadaten- und instanz-basierten Match-Verfahren Mappings zwischen GO-MF und GO-BP berechnet.

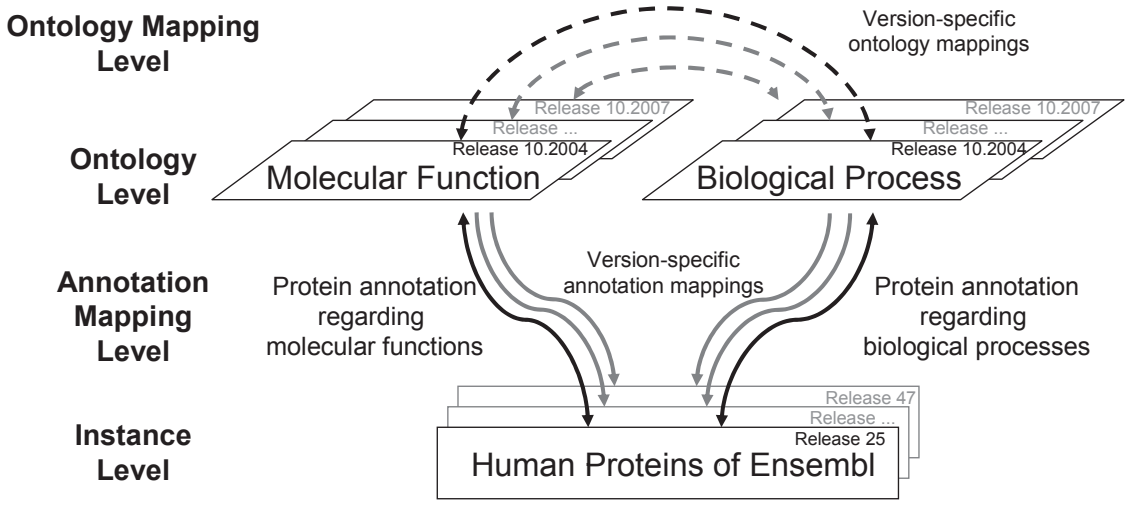


Abbildung 4.2: Analyseszenario für Mappings

4.4.2 Evolution von Instanzen vs. Ontologien

Die Proteine in Ensembl und die zugehörigen Annotationen sowie die beiden Ontologien unterliegen ständigen Änderungen. Das Analyseszenario umfasst 23 Versionen von Ensembl zwischen Dezember 2004 und Oktober 2007 (36 Monate). In Tab. 4.4 werden alle Ensembl Veröffentlichungen inkl. Versionsnummer und Datum

Time		Ensembl Release	NCBI Genome	Used GO Release
2004	Oct.	25	34	02.2004
	Nov.	26	35	
	Dec.	27		09.2004
2005	Feb.	28		
	Mar.	29		
	Apr	30		
	May	31		03.2005
	July	32		
	Sep.	33		
	Oct.	34		
	Nov.	35		
	Dec.	36		
2006	Feb.	37	36	
	Apr.	38		03.2006
	June	39		
	Aug.	40		
	Oct.	41		
	Dec.	42		09.2006
2007	Feb.	43		
	Apr.	44		03.2007
	June	45		05.2007
	Aug.	46		06.2007
	Oct.	47		

Tabelle 4.4: Versionierung von Ensembl

aufgelistet. In den ersten beiden Jahren (2004 und 2005) ist eine aperiodische Veröffentlichung von Versionen zu erkennen. Ab 2006 wird im Schnitt jeden zweiten Monat eine neue Version veröffentlicht. Die Informationen in Ensembl hängen stark von der aktuellen Genom-Assemblierung des NCBI[19] ab. Innerhalb des Analysezeitraums waren die Assemblierungen 34, 35 und 36 gültig. Die letzte Spalte von Tab. 4.4 zeigt, welche GO Version zur Annotation der Proteine verwendet wurde. Es ist zu erkennen, dass Annotationen häufig nicht auf der aktuellen Version von GO beruhen, sondern ältere Versionen referenzieren. So basiert die Ensembl Version 37 von Februar 2006 auf der GO Version von März 2005, d. h. es besteht ein Unter-

[19] `http://www.ncbi.nlm.nih.gov/Genomes/`

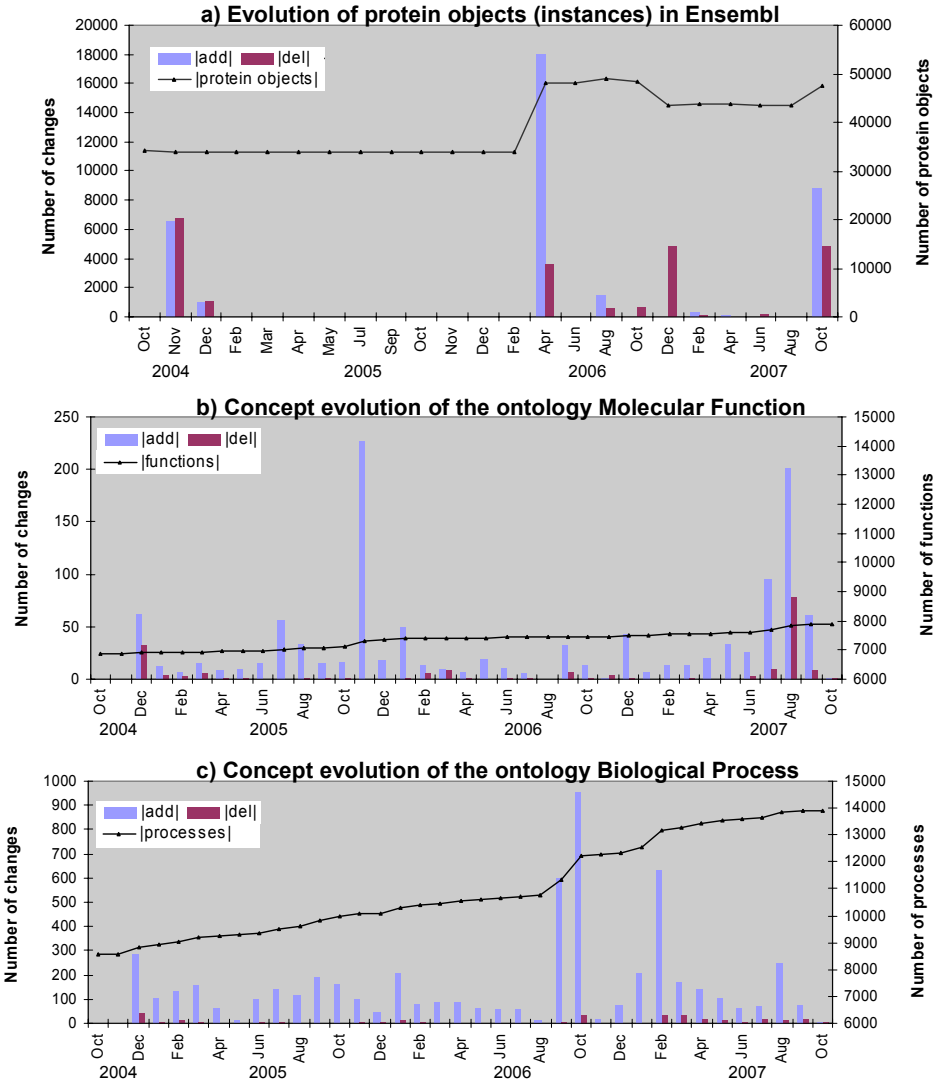


Abbildung 4.3: Evolution von Instanzen (a) und Ontologien (b,c)

schied von 11 Monaten. Die Abstände wurden jedoch im Laufe der Zeit verringert, z.B. Version 45 von Juni 2007 nutzt die GO Version von Mai 2007.

Abb. 4.3 stellt die Evolution der Proteine in Ensembl (Anzahl Proteine, Anzahl eingefügter bzw. gelöschter Proteine) zwischen Oktober 2004 und Oktober 2007 graphisch dar. Insbesondere nach der Veröffentlichung einer neuen Genom-Assemblierung sind massive Änderungen an den Proteinen in Ensembl auszumachen. So resultierten aus dem Wechsel der Genom-Assemblierung von 34 auf 35 zahlreiche Einfügungen und Löschungen in Ensembl, wohingegen die absolute Anzahl an Proteinen nahezu konstant blieb. Beim Wechsel der Genom-Assemblierung von Version 35 auf 36 (April 2006) zeigt die Evolution in Ensembl fünfmal mehr Einfügungen als Löschungen was zu einem sprunghaften Anstieg der absoluten Anzahl an Proteinen auf ca. 16.000 führte. Seit dem Bestehen der Version 36 der Genom-Assemblierung sind in Ensembl vermehrt Änderungen an Proteinen zu beobachten.

Im Vergleich zeigt der untere Teil von Abb. 4.3 zusätzlich die Evolution der beiden Ontologien GO-BP und GO-MF für den gleichen Zeitraum. In beiden Ontologien ist neben einigen Spitzen eine kontinuierliche Evolution mit Einfügungen und Löschungen/Veraltet-Markierungen zu erkennen. Das Wachstum (*growth*-Metrik) zeigt, dass GO-MF im Analysezeitraum den geringsten Zuwachs aufweist ($growth$=1,2). Dieser liegt noch unterhalb dem Wachstum der Proteine in Ensembl ($growth$=1,39). Das stärkste Wachstum ist in GO-BP auszumachen ($growth$=1,74). Zusammengefasst lässt sich feststellen, dass Änderungen in den Ontologien von Einfügungen dominiert werden, Löschungen treten eher selten auf (*adr* von 7 bzw. 16 für GO-MF und GO-BP). In den Proteinen hingegen ist eine signifikant höhere Aktivität bei Löschungen zu beobachten (*adr* von 1,6).

4.4.3 Evolutionsanalyse von Annotation-Mappings

Diese Analyse umfasst die Evolution der beiden Annotation-Mappings zwischen Proteinen und GO-MF bzw. GO-BP. In der Analyse werden für beide Mappings die Ensembl-Versionen 25 (Oktober 2004) und 47 (Oktober 2007) miteinander verglichen. Tab. 4.5 (links) stellt die Evolution der beiden Mappings unter Nutzung der in 4.2.2 eingeführten Metriken dar. Hierzu gehören das Wachstum der Proteine, der Ontologiekonzepte und der Korrespondenzen (*growth*-Metrik) sowie die relative Anzahl eingefügter/gelöschter Elemente ($frac$-Metrik). Des Weiteren wird die relative Abdeckung (*cov*-Metrik) für die an den Mappings beteiligten Instanzquellen und Ontologien angewandt. Die Veränderung der relativen Abdeckung zwischen der ersten und letzten Ensembl-Version wird über die *growth*-Metrik erfasst.

Beide Annotation-Mappings (Protein-MF sowie Protein-BP) zeigen ein ähnliches Evolutionsverhalten. Das Wachstum der absoluten Anzahl von Korrespondenzen ist in beiden Fällen mit 2,82 und 2,47 sehr hoch. Diese Wachstumsraten übersteigen die Raten der Einzelquellen, welche zwischen 1,2 und 1,74 lagen (siehe 4.4.2). Sie

Annotation Mappings	Corresp.		Protein obj.		Concepts	
	growth		growth		growth	
	add-frac	del-frac	add-frac	del-frac	add-frac	del-frac
Protein-MF	2.82		1.99		1.39	
	83%	51%	68%	37%	32%	6%
Protein-BP	2.47		1.90		2.25	
	81%	52%	68%	39%	58%	5%

Annotation Mappings	Protein obj.		Concepts	
	cov_{25}	cov_{47}	cov_{25}	cov_{47}
	$growth_{cov}$		$growth_{cov}$	
Protein-MF	47%	67%	28%	35%
	1.43		1.22	
Protein-BP	43%	59%	20%	26%
	1.36		1.39	

Tabelle 4.5: Wachstum (links) und relative Abdeckung (rechts) für untersuchte Annotation-Mappings

liegen ebenfalls über den Wachstumsraten der annotierten Proteine (*growth* zwischen 1,9 und 1,99) sowie der zur Annotation genutzten Konzepte (*growth* zwischen 1,39 und 2,25). Auch die Häufigkeit von Einfügungen und Löschungen ist größer als die in den Einzelquellen. So enthalten beispielsweise die Annotation-Mappings der letzten Ensembl-Version im Vergleich zur ersten Version 81–83% neu eingefügte Annotationen. Des Weiteren wurden rund 50% der in der ersten Version vorhandenen Annotationen gelöscht. Die Untersuchungen zeigen, dass die Verwendung von Ontologiekonzepten in Annotation-Mappings rasanter steigt als die Anzahl von Ontologiekonzepten bzw. Instanzen in den Einzelquellen selbst. Es ist jedoch auch eine hohe Instabilität aufgrund von Löschungen in den Annotation-Mappings auszumachen.

Die relative Abdeckung in Tab. 4.5 (rechts) untermauert nochmals die vorherigen Erkenntnisse. Das Wachstum an Korrespondenzen führte zu einer erhöhten relativen Abdeckung in den Proteinen. So stieg die relative Abdeckung der Proteine in den Mappings von 43–47% auf 59–67% an, d. h. ein Großteil der Proteine ist nun mit mindestens einem Konzept der GO assoziiert. Ein ähnliches Verhalten ist auch bei den Ontologien erkennbar, denn die relative Abdeckung von GO-MF (GO-BP) verbesserte sich auf 26% (35%).

4.4.4 Evolutionsanalyse von Ontologie-Mappings

Für Ontologie-Mappings wurde eine Evolutionsanalyse für verschiedene Versionen von BP-MF-Mappings durchgeführt. Das BP-MF-Mapping beschreibt, welche Molekularen Funktionen (MF) an welchen Biologischen Prozessen (BP) beteiligt sind. Die manuelle Bestimmung eines solchen Ontologie-Mappings ist gerade für große Ontologien sehr zeitaufwendig. Zudem wurde gezeigt, dass sich Ontologien in den Lebenswissenschaften ständig ändern, was eine regelmäßige manuelle Anpassung bedeuten würde. Deswegen wird in dieser Analyse eine automatische Bestimmung von Ontologie-Mappings angewandt, d. h. über verschiedene Match-Algorithmen werden mögliche Korrespondenzen zwischen den Konzepten beider Ontologien automatisch bestimmt. Es werden vier verschiedene Match-Algorithmen aus [61] innerhalb die-

Ontology Mappings	Corresp.		Mol. Functions		Biol. Processes	
	|C1|-|C2|, grow		grow		grow	
	add-frac	del-frac	add-frac	del-frac	add-frac	del-frac
Base(5)	2780-8973, 3.2		1.8		2.3	
	78%	29%	52%	16%	62%	12%
Min (1.0)	4795-11564, 2.4		1.4		2.1	
	80%	52%	41%	15%	62%	21%
Name (0.5)	5434-15016, 2.8		2.1		1.4	
	77%	36%	57%	10%	44%	20%
Name (0.7)	389-592, 1.5		1.3		1.3	
	45%	17%	32%	12%	34%	15%

Ontology Mappings	Mol. Functions		Biol. Processes	
	cov_{25}	cov_{47}	cov_{25}	cov_{47}
	$grow_{cov}$		$grow_{cov}$	
Base(5)	7%	12%	6%	8%
	1.7		1.3	
Min (1.0)	23%	30%	17%	20%
	1.3		1.2	
Name (0.5)	25%	47%	18%	15%
	1.9		0.8	
Name (0.7)	5%	6%	4%	3%
	1.2		0.7	

Tabelle 4.6: Wachstum (links) und relative Abdeckung (rechts) für generierte Ontologie-Mappings zwischen GO-MF und GO-BP

ser Analyse untersucht. Die ersten beiden Ansätze sind instanz-basiert und beruhen auf der Idee, dass zwei Konzepte sich stark ähneln, falls sie eine hinreichend große Anzahl gemeinsamer Instanzen besitzen. In dieser Analyse werden somit zwei Konzepte aus GO-MF und GO-BP als ähnlich angesehen, falls es Proteine gibt, welche sowohl das eine als auch das andere Konzept annotieren. Beim Ansatz *Base(5)* bilden zwei Konzepte eine Korrespondenz, falls diese mindestens fünf gemeinsame Proteine aufweisen. *Min(1.0)* hingegen verwendet eine Min-Ähnlichkeit mit Schwellwert 1,0, d. h. zwei Konzepte bilden ein Paar falls alle Proteine des Konzepts mit der geringeren Anzahl an Annotationen ebenfalls mit dem anderen Konzept annotiert sind. Die beiden anderen Ansätze sind metadaten-basiert und nutzen die Ähnlichkeit von Konzeptnamen aus. Es wird angenommen, dass eine Korrespondenz zwischen zwei Konzepten vorliegt, falls die String-Ähnlichkeit ihrer Namen (Trigram) einen vorgegebenen Schwellwert überschreitet. Es werden die beiden Schwellwerte 0,5 und 0,7 unterschieden, die beiden zugehörigen Verfahren werden mit *Name(0.5)* und *Name(0.7)* bezeichnet. Mit Hilfe der vier dargestellten Verfahren wurden BP-MF-Mappings für die Ontologieversionen von Februar 2004 (innerhalb von Ensembl 25) und Juni 2007 (innerhalb von Ensembl 47) berechnet.

Tab. 4.6 (links) zeigt das Wachstum der Ontologie-Mappings zwischen den beiden Versionen sowie die relativen Anteile eingefügter bzw. gelöschter Korrespondenzen. In der Spalte „Corresp." wird die absolute Anzahl von Korrespondenzen für die alte und neue Version dargestellt, z. B. stieg die Anzahl an Korrespondenzen bei *Base(5)* von 2.780 auf 8.973 an, was einem Wachstum von 3,2 entspricht. Tab. 4.6 (rechts) stellt die entsprechenden relativen Abdeckungen der beiden Ontologien dar, d. h. wie stark ist eine Ontologie an einem Ontologie-Mapping beteiligt. Beispielsweise stieg die relative Abdeckung im Falle von *Base(5)* für GO-MF von 7% auf 12% an.

Die generierten Ontologie-Mappings unterscheiden sich signifikant in ihrer Evolution. Für *Name* hängt die Anzahl der Ergebniskorrespondenzen erheblich von der Wahl des Schwellwertes ab. Ein niedriger Schwellwert von 0,5 erzeugt viele Korre-

spondenzen, was zu einer hohen relativen Abdeckung führt. Hierbei besteht jedoch das Risiko viele falsch-positive Korrespondenzen im Endergebnis zu erhalten. Umgekehrt führt die Anwendung eines höheren Schwellwerts (0,7) zu einem deutlich reduzierten Ergebnisumfang mit entsprechend geringerer relativer Abdeckung in den Ontologien. Ebenfalls ist zu erkennen, dass ein höherer Schwellwert deutlich stabilere Ontologie-Mappings produziert. Beispielsweise besitzt *Name(0.7)* mit 17% den geringsten Anteil gelöschter Korrespondenzen. Interessanterweise nahm die relative Abdeckung in GO-BP für die *Name*-Algorithmen ab. Vermutlich ist GO-BP so stark gegenüber GO-MF gewachsen (siehe 4.3.1), dass für viele Konzepte in GO-BP kein entsprechendes Konzept in GO-MF gefunden werden konnte.

Die beiden instanz-basierten Match-Algorithmen erzielten eine relativ große Menge an Ergebniskorrespondenzen und weisen einen enorm starken Zuwachs zwischen den beiden Versionen (*growth* zwischen 2,4 und 3,2) auf, d. h. die Mappings wachsen stärker als die am Mapping beteiligten Ontologien. Das Wachstum in den Annotation-Mappings setzt sich somit auch in einem Wachstum der Ontologie-Mappings fort. Der *Base(5)*-Algorithmus erzielt stabilere Ergebnisse als *Min(1.0)*. So liegt beispielsweise der Anteil gelöschter Korrespondenzen mit 29% für *Base(5)* deutlich unter dem Wert von 52% für *Min(1.0)*. Umgekehrt besitzt das *Min*-Ergebnis eine höhere relative Abdeckung in beiden Ontologien.

4.5 Zusammenfassung

Das Kapitel stellte ein generelles Framework zur quantitativen Evolutionsanalyse von Ontologien und Ontologie-basierten Mappings vor. Das Framework wurde für eine umfassende Analyse der Evolution von 16 Ontologien der Lebenswissenschaften zwischen 2004 und 2008 verwendet. Die Analyse zeigte, dass alle untersuchten Ontologien ständig verändert (angepasst) werden und ein signifikantes Wachstum aufweisen. Die häufigste Art von Änderungen sind Einfügungen, jedoch wurden auch zahlreiche Konzepte gelöscht oder auf veraltet („obsolete“) gesetzt. Der „Obsolete“-Mechanismus wird in nahezu allen untersuchten Ontologien eingesetzt, um direkte Löschungen zu vermeiden. Dadurch wird die Stabilität der Ontologien erhöht und eine Migration auf neuere Versionen erleichtert. is_a Beziehungen dominieren derzeit die Struktur der untersuchten Ontologien (mehr als 85% Anteil), jedoch ist in den letzten Jahren ein Anstieg in den Anteilen von part_of und sonstigen Beziehungen zu erkennen. Des Weiteren ist eine Zunahme in der Komplexität der inneren Strukturen der Ontologien auszumachen (z. B. Anteil innerer Konzepte, Pfadlängen, Anzahl von Pfaden). Dies zeigt, dass die Ontologien mehr und mehr strukturiertes Wissen aufnehmen und eine stetige Verfeinerung stattfindet.

Das Framework wurde ebenfalls zur quantitativen Evolutionsanalyse von Protein-Instanzen, Annotation- und Ontologie-Mappings verwendet. So wurde in der Datenquelle Ensembl ein enormes Wachstum in der Anzahl von Protein-Annotationen

zu Konzepten der Gene Ontology festgestellt. Die Instabilität in den Protein-Instanzen mit teilweise enormen Löschraten führte zu einer erhöhten Instabilität in den Annotation-Mappings. Zur Untersuchung der Evolution von Ontologie-Mappings wurden einige metadaten- bzw. instanz-basierte Match-Algorithmen zur automatischen Bestimmung von Mappings zwischen den GO Subontologien evaluiert. Die generierten Ontologie-Mappings unterliegen einer stärkeren Evolution als die verwendeten Ontologien, dies gilt insbesondere für die instanz-basierten Match-Algorithmen. Metadaten-basierte Verfahren (z. B. auf Basis der Namensähnlichkeit von Konzepten) weisen dagegen eine verbesserte Stabilität auf. Der Grund hierfür liegt in der Abhängigkeit der instanz-basierten Verfahren von Änderungen in den Instanzen und Annotation-Mappings.

Die Erkenntnisse aus den durchgeführten Analysen bilden die Basis für weitere Arbeiten und Untersuchungen. So wurde der in Abschnitt 4.2.1 eingeführte vereinfachte Diff zwischen Ontologieversionen erweitert, um zusätzlich komplexe Änderungen zu identifizieren (siehe Diff in Kapitel 5). Der Frage, in welchen Teilen einer Ontologie besonders häufig Änderungen stattfinden, wird mit einem gesonderten Algorithmus entgegnet (siehe Kapitel 6). Die auf dem Framework basierte Webapplikation OnEX zur ad-hoc Evolutionsanalyse von Ontologien wird in Kapitel 7 beschrieben. Ergebnisse der Evolutionsanalyse für Mappings führten zur Konzeption und Umsetzung von Verfahren zur Erkennung stabiler Korrespondenzen für Annotation-Mappings [38] und Ontologie-Mappings [116]. Die beiden zuletzt genannten Arbeiten werden innerhalb dieser Dissertation nicht näher betrachtet.

5

Differenzbestimmung (DIFF) zwischen Ontologieversionen

5.1 Motivation

Die enorm gestiegene Verwendung von Ontologien in verschiedenen Domänen wie den Lebenswissenschaften oder dem Web erfordert eine ständige Aktualisierung und Anpassung an veränderte Anforderungen, um aktuelle sowie korrekte Informationen anbieten zu können. So müssen beispielsweise Produktkataloge angepasst werden, um neuartige Produkte aufzunehmen, veraltete Produkte zu entfernen oder gewisse Produktkategorien besser zu vermarkten. Auch Webverzeichnisse (z. B. DMoz[20] oder Google Directory[21]) werden verändert, so dass stets ein hochaktueller Wissensstand abrufbar ist. Da viele Applikationen wie z. B. Suchmaschinen oder Empfehlungsdienste von den entsprechenden Ontologien abhängig sind, sind bei Veränderungen in den Ontologien (Veröffentlichung einer neuen Ontologieversion) häufig auch Anpassungen in den Ontologie-nutzenden Daten, Applikationen und Systemen notwendig.

Das folgende Kapitel beschäftigt sich mit der Problematik der Bestimmung eines sogenannten Diff Evolution-Mappings zwischen zwei Versionen einer Ontologie. Das Ziel dabei ist es, die Änderungen zu bestimmen, welche angewandt auf die alte Ontologieversion die neue Ontologieversion erzeugen können. Ein solches Evolution-

[20] http://www.dmoz.org
[21] http://www.google.com/dirhp

Mapping ist in vielerlei Hinsicht informativ und wertvoll. So erfahren Anwender und Nutzer von Ontologien, wie sich eine alte Ontologieversion gegenüber der neu veröffentlichten Version verändert hat. Weiterhin besteht die Möglichkeit abzuschätzen, ob beispielsweise eine Anwendung angepasst werden muss oder ob Daten migriert werden müssen, um kompatibel mit der neuen Ontologieversion zu sein. Falls eine solche Migration nötig ist, können Informationen über Änderungen aus dem Evolution-Mapping Aufschluss darüber geben, wie die betroffenen Daten migriert werden können.

Häufig wird zwischen zwei Ontologieversionen ein relativ einfaches Diff Evolution-Mapping bestehend aus *add* und *del* Änderungen an Ontologieelementen wie Konzepten, Beziehungen oder Attributen berechnet (siehe generisches jedoch vereinfachtes Evolutionsmodell in Kapitel 4). Ein solches Mapping besitzt jedoch nur eine sehr einfache Sicht auf die Evolution zwischen zwei Versionen. Zwar ist ein solches Evolution-Mapping relativ einfach berechenbar, allerdings ist die Nutzbarkeit insbesondere bei großen Ontologien wie in den Lebenswissenschaften sehr eingeschränkt. Insbesondere menschliche Nutzer müssen oftmals mit einer riesigen Menge einfacher Änderungen umgehen und können leicht den Überblick über die Evolution verlieren. Die Frage „Was hat sich zwischen zwei Ontologieversionen verändert?“ wird nur auf einem semantisch niedrigen Niveau beantwortet und die Ergebnisse sind somit nur eingeschränkt in weiteren Anwendungen nutzbar. Aus den genannten Gründen erscheint es sinnvoll ein semantisch reichhaltigeres Diff Evolution-Mapping zu bestimmen. Komplexe Änderungsoperationen wie das Zusammenfassen (merge) von Konzepten in ein Konzept oder das Aufspalten (split) eines Konzepts in mehrere Konzepte besitzen eine höhere Ausdrucksstärke und Verbessern somit das Verständnis der Evolution zwischen zwei Ontologieversionen. Gerade bei großen Ontologien geben Änderungsoperationen wie beispielsweise das Hinzufügen oder Löschen ganzer Subgraphen einen kompakten jedoch detaillierten Einblick in die Ontologieevolution. Darüber hinaus ist ein semantisch reichhaltiges Evolution-Mapping ebenfalls für Ontologieentwickler selbst interessant. Falls diese eine Menge einfacherer Änderungen an einer Ontologie durchgeführt haben, können sie mit Hilfe eines semantisch reichhaltigeren Evolution-Mappings nachvollziehen, welche Auswirkungen die Menge ihrer einfachen Änderungen zur Folge hatte.

Als ein motivierendes Beispiel dient die Evolution eines Produktkatalogs für Speichermedien dargestellt in Abb. 5.1. Das Ziel ist es, ein Diff Evolution-Mapping zwischen den beiden Ontologieversionen (alte Version links, neue Version rechts) des Katalogs zu bestimmen. Ein einfacher Ansatz, welcher lediglich eine *add*/*del* Semantik für Ontologieelemente unterstützt, würde u.a. eine Löschung der Kategorien 'DVD-ROM' und 'CD-RW' feststellen. Jedoch wurden diese Kategorien der alten Version in der Kategorie 'Other' zusammengefasst (merge). Aus diesem Grunde wird nachfolgend eine Match-basierte Erzeugung von Diff Evolution-Mappings verfolgt. Die in Abb. 5.1 weiß hinterlegten Kategorien der alten und neuen Version haben sich nicht verändert. Es wird angenommen, dass die Korrespondenzen zwischen die-

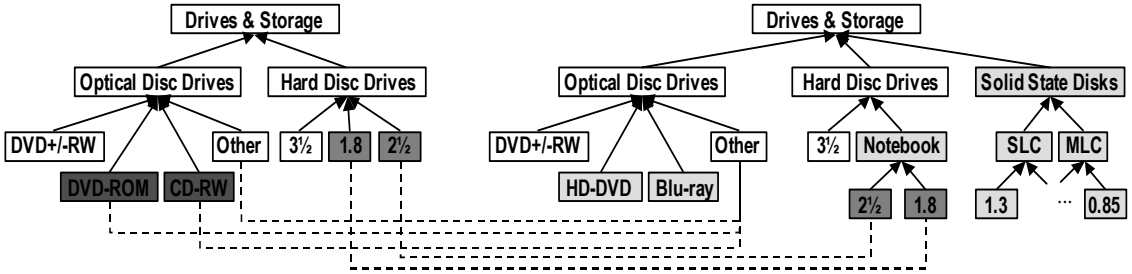


Abbildung 5.1: Motivierendes Beispiel zur Bestimmung des Diffs zwischen zwei Ontologieversionen

sen Kategorien Teil des Matchergebnisses sind. Gestrichelte Linien kennzeichnen weitere Korrespondenzen im Matchergebnis zwischen Kategorien der alten und neuen Ontologieversion, welche für das spätere Diff Evolution-Mapping relevant sind. Die vorliegenden Korrespondenzen zwischen 'DVD-ROM' und 'Other' bzw.'CD-RW' und 'Other' werden beispielsweise verwendet, um die Zusammenfassung der Konzepte 'DVD-ROM', 'CD-RW' und 'Other' zu 'Other' (merge) zu bestimmen. Eine weitere komplexe Änderung umfasst das Einfügen einer Subontologie mit der Wurzel 'Solid State Disks'. Eine solche kompakte Darstellung ist hilfreicher und intuitiv verständlicher als die pure Aufzählung der Einfügungen der einzelnen Kategorien der Subontologie. Ebenfalls ist es sinnvoll, eine Verschiebung (move) der Kategorien '1.8' und '$2^1/_2$' von 'Hard Disk Drives' nach 'Notebook' im finalen Diff Evolution-Mapping zu erfassen.

Im nachfolgenden wird ein regelbasierter Ansatz zur automatischen Bestimmung komplexer Änderungen innerhalb eines Diff Evolution-Mappings vorgestellt. Die wesentlichen Beiträge sind:

- Es wird ein Modell für ein invertierbares Diff Evolution-Mapping zwischen zwei Ontologieversionen eingeführt. Hierzu werden sowohl einfache als auch komplexe Änderungsoperationen definiert.
- Es wird ein generischer, regelbasierter Diff Algorithmus vorgestellt, welcher die Bestimmung eines Diff Evolution-Mappings zwischen zwei Ontologieversionen ermöglicht. Der Algorithmus verwendet das Matchergebnis (Menge einfacher 1:1 Korrespondenzen) eins Match zwischen den Ontologieversionen. Das Diff Evolution-Mapping wird automatisch durch die Anwendung sogenannter COG-Regeln (Change Operation Generating rules) erzeugt. Es wird die Korrektheit sowie die Komplexität des Algorithmus gezeigt. Des Weiteren wird ein Algorithmus zur Transformation der Ontologieversionen auf Basis des bestimmten Diff Evolution-Mappings präsentiert.
- Der Ansatz ermöglicht ebenfalls die Bestimmung eines inversen Diff Evolution-Mappings, welches aus den inversen Änderungsoperationen besteht. Ein sol-

ches inverses Mapping kann verwendet werden, um ältere Ontologieversionen wiederherzustellen („undo-Funktionalität“).

- Der vorgestellte Algorithmus wurde implementiert und erfolgreich auf Realwelt-Ontologien angewandt. Es wurden Diff Evolution-Mappings für Versionen der Gene Ontology sowie für ein Webverzeichnis bestimmt und evaluiert.

5.2 Änderungsoperationen und Diff Evolution-Mappings

Im folgenden werden zunächst Änderungsoperationen erläutert und klassifiziert, anschließend erfolgt die Definition von Match- und Evolution-Mappings sowie eine abschließende Beschreibung der Problemstellung. Für das zugrunde liegende Ontologiemodell sowie die Versionierung wird auf das Grundlagenkapitel (Kapitel 3) verwiesen.

5.2.1 Änderungsoperationen

Es werden grundsätzlich zwei Mengen von Änderungsoperationen unterschieden: einfache Änderungen bzw. Basisänderungen (Menge B_{op}) und komplexe Änderungen (Menge C_{op}). Die nachfolgend vorgeschlagenen Änderungsoperationen für beide Mengen werden durch die derzeitige Implementierung unterstützt, können jedoch je nach Anforderungen entsprechend erweitert und angepasst werden.

Basisänderungen (erfasst innerhalb der Menge B_{op}) beziehen sich auf ein einzelnes Konzept, eine Beziehung oder ein Attribut und können eine Einfügung (*add*), eine Löschung (*del*) oder eine Verknüpfung (*map*) darstellen. Demnach werden die folgenden neun Basisänderungsoperationen unterschieden:

- $mapC(c1, c2)$: Verknüpfung zwischen einem Konzept $c1$ der ersten Version und einem Konzept $c2$ der zweiten Version
- $addC(c)$: Einfügen eines Konzepts c in die Ontologie
- $delC(c)$: Löschen eines Konzepts c aus der Ontologie
- $mapR(r1, r2)$: Verknüpfung zwischen einer Beziehung $r1$ der ersten Version und einer Beziehung $r2$ der zweiten Version
- $addR(r)$: Einfügen einer Beziehung r in die Ontologie
- $delR(r)$: Löschen einer Beziehung r aus der Ontologie

- $mapA(a1, a2)$: Verknüpfung zwischen Attribut $a1$ der ersten Ontologie und einem Attribut $a2$ der zweiten Ontologie
- $addA(a)$: Einfügen eines Attributs a in die Ontologie
- $delA(a)$: Löschen eines Attributs a aus der Ontologie

Im weiteren Verlauf werden diese Basisänderungen verwendet, um eine Migration zwischen Ontologieversionen zu realisieren. Beispielsweise wird ein neues Konzept mit *addC* in die Ontologie eingefügt und über *addR* wird seine Position innerhalb der Ontologiestruktur festgelegt.

Die komplexen Änderungsoperationen werden innerhalb der Menge C_{op} zusammengefasst. Sie bauen auf den Basisänderungsoperationen oder anderen komplexen Änderungsoperationen auf und erfassen Änderungen somit auf einem semantisch höheren Level. Jedoch kann der Effekt einer komplexen Änderung mit Hilfe einer Menge von Basisänderungen realisiert werden. Diese Tatsache wird später genutzt, um Ontologieversionen zu migrieren. Dabei soll aus einer Ontologieversion unter Nutzung des Diff Evolution-Mappings die veränderte Ontologieversion erzeugt werden (siehe 5.4.5).

Einige komplexe Änderungsoperationen beziehen sich auf einzelne Konzepte (dargestellt durch Kleinbuchstaben). Es werden die folgenden Operationen betrachtet:

- $substitute(c1, c2)$: Ersetzung des Konzepts $c1$ durch $c2$
- $move(c, c\_from, c\_to)$: Verschiebung eines Konzepts c und dessen Subgraphen von c_from nach c_to
- $toObsolete(c)$: Setzen des Status eines Konzepts c auf veraltet, d. h. das Konzept soll nicht mehr aktiv verwendet werden
- $revokeObsolete(c)$: Zurücksetzen des Status eines Konzepts c, d. h. das Konzept kann wieder aktiv verwendet werden

Die beiden letzten Änderungsoperationen werden insbesondere in Ontologien der Lebenswissenschaften angewandt, um den Aktiv-Zustand eines Konzepts festzulegen. Inaktive (veraltete) Konzepte sollen innerhalb von Anwendungen oder Analysen nicht mehr verwendet werden, da sie veraltetes Wissen darstellen. Zu Dokumentationszwecken sind die Konzepte jedoch weiterhin Bestandteil der Ontologie. Die Berücksichtigung solcher spezieller Änderungsoperationen unterstreicht die Flexibilität des Ansatzes, d. h. es können je nach Domäne oder Anwendung spezielle Änderungsoperationen definiert und bestimmt werden.

Alle anderen komplexen Änderungsoperationen beziehen mehrere Ontologieelemente (dargestellt mit Großbuchstaben) ein. Es werden die folgenden Änderungsoperationen unterschieden:

- $addLeaf(c, C\_Parents)$: Einfügen eines Blattkonzepts c unterhalb der Elternkonzepte $C\_Parents$

- $delLeaf(c, C\_Parents)$: Löschen eines Blattkonzepts c unterhalb der Elternkonzepte $C\_Parents$

- $merge(Source\_C, target\_c)$: Zusammenfassen mehrerer Konzepte $Source\_C$ in einem Konzept $target\_c$

- $split(source\_c, Target\_C)$: Aufspalten eines Konzepts $source\_c$ in mehrere Konzepte $Target\_C$

- $addSubGraph(c\_root, C\_Sub)$: Einfügen eines Subgraphen mit Wurzel c_root und den Konzepten C_Sub

- $delSubGraph(c\_root; C\_Sub)$: Löschen eines Subgraphen mit Wurzel c_root und den Konzepten C_Sub

So kann beispielsweise das Zusammenfassen der Kategorien 'DVD-ROM', 'CD-RW' und 'Other' aus dem motivierenden Beispiel in Abb. 5.1 wie folgt beschrieben werden: *merge*({'DVD-ROM','CD-RW','Other'},Other).

Wie bereits erwähnt, können komplexe Änderungsoperationen durch eine Menge von Basisoperationen realisiert werden. Aus diesem Grund werden innerhalb der regelbasierten Bestimmung komplexer Änderungen (siehe Algorithmus in 5.4) Abhängigkeiten zwischen Änderungsoperationen verwaltet. So ist es möglich nachzuvollziehen, welche einfachen Änderungsoperationen eine komplexe Änderungsoperation ergeben. Somit wird die Implementierung der komplexen Änderungsoperationen durch eine Menge einfacherer Basisoperationen sichergestellt. Für die komplexe Änderungsoperation *merge* aus dem Beispiel ergeben sich die folgenden abhängigen *mapC* Basisoperationen: *mapC*('DVD-ROM','Other'), *mapC*('CD-RW','Other') und *mapC*('Other','Other'). Mit Hilfe dieser drei Änderungsoperationen kann die komplexe Operation *merge*({'DVD-ROM','CD-RW','Other'},Other) implementiert werden.

Jede Änderungsoperation besitzt eine inverse Änderungsoperation, welche den Effekt einer Änderung rückgängig macht. Die Inverse einer Änderungsoperation *chgOp* ist eine andere Änderungsoperation mit veränderten (permutierten) Parametern. Zum Beispiel gehört zu *merge*({'DVD-ROM','CD-RW','Other'},Other) die inverse Änderungsoperation *split*('Other',{'DVD-ROM','CD-RW','Other'}), d. h. die Kategorie 'Other', in welche die Einzelkategorien zusammenfasst wurden, wird in der Umkehroperation in eben diese Einzelkategorien aufgespalten. Diese Symmetrie erlaubt es zu jeder Änderungsoperation die zugehörige Umkehroperation zu bestimmen. Im Anhang B.1 werden alle Änderungsoperationen und ihre Inversen aufgelistet.

5.2.2 Match- und Evolution-Mappings

Im Weiteren werden Änderungen zwischen zwei Ontologieversionen O_{old} und O_{new} mittels eines Mappings repräsentiert. Im Model Management [11, 12, 78] verbindet ein Mapping $map(O_{old}, O_{new})$ Elemente der alten Ontologieversion O_{old} mit Elementen der neuen Ontologieversion O_{new}. Nachfolgend wird zwischen Match-Mappings $match(O_{old}, O_{new})$ und Evolution-Mappings $diff(O_{old}, O_{new})$ unterschieden. Ein Match-Mapping repräsentiert semantische Korrespondenzen zwischen zwei Ontologieversionen und verbindet unveränderte sowie veränderte jedoch zusammengehörige Ontologieelemente. Es wird angenommen, dass Match-Mappings lediglich aus einfachen 1:1 Korrespondenzen zwischen Konzepten bestehen, d. h. ein Match-Mapping kann formal wie folgt beschrieben werden: $match(O_{old}, O_{new}) = \{matchC(c1, c2) | c1 \in O_{old}, c2 \in O_{new}\}$.

Im Gegensatz zu einem Match-Mapping deckt ein Evolution-Mapping die Unterschiede zwischen den Ontologieversionen ab, d. h. es beinhaltet alle Änderungen zwischen den beiden Versionen inklusive Einfügungen und Löschungen. Unveränderte Ontologieelemente, welche innerhalb eines Match-Mappings erfasst werden, sind somit kein Bestandteil eines Evolution-Mappings. Allgemein können Evolution-Mappings die zuvor (siehe 5.2.1) beschriebenen Änderungsoperationen *chgOp* beinhalten. So ist ein Evolution-Mapping *diff* zwischen zwei Ontologieversionen O_{old} und O_{new} formal wie folgt definiert: $diff(O_{old}, O_{new}) = \{chgOp(\ldots) | chgOp \in B_{Op} \cup C_{Op}\}$. Ein Evolution-Mapping sollte vollständig sein, d. h. es sollte alle Änderungen zwischen O_{old} und O_{new} erfassen. Dadurch ist es es möglich, die geänderte Version O_{new} auf Basis der alten Version O_{old} und mit Hilfe der Änderungen im Evolution-Mapping $diff(O_{old}, O_{new})$ zu erzeugen.

Die einfache Variante eines Diff Evolution-Mappings $diff_{basic}$ beinhaltet lediglich Basisänderungsoperationen, d. h. *add*, *del* und *map* Änderungen. Somit wird ein einfaches Evolution-Mapping formal wie folgt definiert: $diff_{basic} = \{chgOp(\ldots) | chgOp \in B_{Op}\}$. Das Ziel des Ansatzes besteht jedoch darin neben dem einfachen auch ein semantisch reichhaltigeres und ausdrucksstärkeres Evolution-Mapping (mit komplexen Änderungsoperationen) zu bestimmen. Dieses finale Evolution-Mapping $diff_{compact}$ sollte daher nur noch die semantisch ausdrucksstarken Änderungsoperationen beinhalten. Es liegt nahe, dass $diff_{compact}$ generell weniger Änderungsoperationen als das einfache Evolution-Mapping $diff_{basic}$ besitzt, da oftmals eine Menge einfacherer Änderungsoperationen durch eine komplexe Änderungsoperation ersetzt werden kann.

Für Evolution-Mappings $diff(O_{old}, O_{new})$ soll gleichermaßen ein zugehöriges inverses Evolution-Mapping bestimmt werden. Dieses inverse Evolution-Mapping kann u.a. zur Migration der neuen Ontologieversion O_{new} in die alte Ontologieversion O_{old} verwendet werden. Im Weiteren Verlauf wird gezeigt, wie auf Basis der Inversen einer Änderungsoperation ein solches inverses Evolution-Mapping abgeleitet werden kann (siehe 5.4.6).

5.2.3 Problemstellung

Das folgende Problem wird in diesem Kapitel näher untersucht. Für zwei gegebene Ontologieversionen O_{old} und O_{new} derselben Ontologie O und einem Match-Mapping $match(O_{old}, O_{new})$ besteht die Aufgabe darin, ein einfaches Diff Evolution-Mapping $diff_{basic}(O_{old}, O_{new})$ sowie ein semantisch ausdrucksstarkes Evolution-Mapping $diff_{compact}(O_{old}, O_{new})$ zu bestimmen. Ferner sollen die zugehörigen inversen Evolution-Mappings bestimmt werden. Der zugrunde liegende Diff Algorithmus muss in der Lage sein, alle definierten Änderungsoperationen (siehe 5.2.1) zu erkennen. Zudem sollten die berechneten Evolution-Mappings wie auch ihre Inversen vollständig sein, d. h. sie sollten alle Änderungen zwischen den beiden Ontologieversionen enthalten. Für $diff_{basic}$ sollte darüber hinaus die Menge der Änderungsoperationen minimal sein, d. h. das Mapping sollte nur die einfachen Änderungsoperationen enthalten, welche für eine korrekte Migration von O_{old} nach O_{new} notwendig sind. Das $diff_{compact}$ Evolution-Mapping sollte bzgl. der definierten komplexen Änderungsoperationen minimal sein. Ferner soll die Inverse des Evolution-Mappings eine Erzeugung der alten Ontologieversion auf Basis der neuen Ontologieversion ermöglichen. Des Weiteren soll der Algorithmus skalierbar sein, d. h. auch für große Ontologien anwendbar sein.

5.3 Regelbasierte Erkennung von Änderungen

Die Erkennung der einfachen und komplexen Änderungsoperationen basiert auf sogenannten *Change Operation Generating Rules* (COG Regeln). Jede COG Regel besteht aus einer Menge von Vorbedingungen, welche in Prädikatenlogik formuliert werden. Falls alle definierten Vorbedingungen einer Regel erfüllt sind, wird eine Sequenz von Aktionen (oder Einzelaktionen) ausgelöst. Eine Aktion kann entweder Änderungsoperationen erzeugen (**create**) oder existierende Operationen, welche nicht mehr benötigt werden, löschen (**eliminate**). Es werden grundsätzlich drei Typen von COG Regeln unterschieden: (1) Basic, (2) Complex und (3) Aggregation COG Regeln. Während die ersten beiden zur Bestimmung der einfachen bzw. komplexen Änderungsoperationen dienen, werden die Aggregation Regeln für das Zusammenfassen komplexer Änderungsoperationen verwendet.

Wie bereits angesprochen, können COG Regeln nicht nur neue Änderungsoperationen erzeugen, sondern auch redundante, nicht mehr benötigte Änderungsoperationen aus einem Evolution-Mapping löschen. Die Regeln werden im Algorithmus (siehe 5.4) verwendet, um ein einfaches sowie ein ausdrucksstarkes Evolution-Mapping für zwei Ontologieversionen zu bestimmen. Der Ansatz ist flexibel und generisch, da neue COG Regeln einfach hinzufügt oder angepasst werden können. So können u. a. veränderte Anforderungen wie beispielsweise die Erkennung neuartiger Änderungsoperationen zeitnah umgesetzt werden. Auch auf Spezifika von Ontologien (z. B.

Umsetzung des „Obsolete“-Mechanismus) kann mit speziellen COG Regeln reagiert werden.

Die nachfolgenden Unterabschnitte stellen die verschiedenen Typen von COG Regeln detaillierter vor. Zugehörige Beispiele illustrieren die Arbeitsweise der Regeln. Die komplette Liste aller verwendeten COG Regeln kann im Anhang B.2 eingesehen werden. In den Regeldefinitionen werden Einzelelemente wie ein Konzept oder eine Beziehung stets mit Kleinbuchstaben ($a, b, \ldots \in O$) und Elementmengen mit Großbuchstaben ($A, B, \ldots \subseteq O$) gekennzeichnet. Jede dargestellte COG Regel besitzt eine eindeutige Identifikationsnummer, welche gleichzeitig den Typ der Regel inkludiert ($b1, b2, \ldots$ für Basic, $c1, c2, \ldots$ für Complex und $a1, a2, \ldots$ für Aggregation COG Regeln). Da Regeln auf Basis existierender Änderungsoperationen in einem Evolution-Mapping weitere (komplexere) Änderungsoperationen bestimmen, liegt eine Abhängigkeit zwischen den Regeln vor, d. h. COG Regeln müssen in einer vordefinierten Reihenfolge angewandt werden (siehe 5.4), um ein korrektes Evolution-Mapping zu erzeugen. Die Identifikationsnummern geben dabei an, welche Regel vor einer anderen Regel ausgeführt werden muss, z. B. Regel (b_{10}) vor Regel (b_{11}).

5.3.1 Basic COG Regeln

Basic COG Regeln (b-COG) verwenden Informationen aus dem Match-Mapping sowie den beiden Ontologieversionen, um einfache Änderungen zu bestimmen. Die folgenden fünf b-COG Regeln werden zur Berechnung von *addC*, *delC* und *mapC* Änderungen verwendet:

(b_1) $c \in O_{new} \wedge \nexists a(a \in O_{old} \wedge matchC(a, c)) \rightarrow \textbf{create}[addC(c)]$

(b_2) $c \in O_{old} \wedge \nexists a(a \in O_{new} \wedge matchC(c, a)) \rightarrow \textbf{create}[delC(c)]$

(b_3) $a \in O_{old} \wedge b \in O_{new} \wedge matchC(a, b) \wedge a \neq b \rightarrow \textbf{create}[mapC(a, b)]$

(b_4) $a \in O_{old}, O_{new} \wedge matchC(a, a) \wedge \exists b(b \in O_{new} \wedge matchC(a, b) \wedge a \neq b)$
$\rightarrow \textbf{create}[mapC(a, a)]$

(b_5) $a \in O_{old}, O_{new} \wedge matchC(a, a) \wedge \exists b(b \in O_{old} \wedge matchC(b, a) \wedge a \neq b)$
$\rightarrow \textbf{create}[mapC(a, a)]$

Für das Beispiel aus Abb. 5.1 würde Regel (b_1) *addC* Änderungen wie beispielsweise *addC*('Blu-ray') oder *addC*('HD-DVD') erzeugen. Regel (b_3) generiert *mapC* Änderungen für veränderte Konzepte, z. B. *mapC*('CD-RW','Other'). Des Weiteren werden die Regeln (b_4) und (b_5) verwendet, um *mapC* Änderungen für Konzepte mit Mehrfachkorrespondenzen zu bestimmen. Die Beteiligung eines Konzepts an mehreren Korrespondenzen impliziert eine geänderte Semantik für das Konzept, d. h.

diese Art von Änderungen werden ebenfalls in das Evolution-Mapping aufgenommen. So wird im Beispiel eine Änderung *mapC*('Other','Other') generiert, da die beiden Konzepte 'CD-RW' und 'DVD-ROM' ebenfalls Korrespondenzen mit 'Other' aufweisen. Anhang B.2 listet weitere b-COG Regeln zum Erkennen von Änderungen an Beziehungen und Attributen von Konzepten. b-COG Regeln werden später im Algorithmus (siehe 5.4.3) zur Erzeugung des Basis-Evolution-Mapping $diff_{basic}$ eingesetzt.

5.3.2 Complex COG Regeln

Complex COG Regeln (c-COG) sind nicht-rekursiv und erlauben die Erzeugung von komplexen Änderungen basierend auf einfachen und/oder anderen komplexen Änderungen. Komplexe Änderungen operieren auf Mengen von Ontologieelementen und werden in einem zweistufigen Verfahren abgeleitet. Zunächst werden c-COG Regeln verwendet, um komplexe Änderungen, welche lediglich einzelne Ontologieelemente umfassen, generiert. Danach erfolgt eine Aggregation (siehe Aggregation COG Regeln) dieser zu Änderungen mit mengenwertigen Parametern (d. h. Mengen von Ontologieelementen). Beispielsweise werden so für die komplexe *merge* Operation zuerst partielle *merge* Operationen mit Einzelelementen erzeugt. Die entsprechende c-COG Regel sieht wie folgt aus:

$$(c_7)\ \ a, b \in O_{old} \wedge c \in O_{new} \wedge mapC(a,c) \wedge mapC(b,c) \wedge a \neq b \wedge \nexists d(d \in O_{new} \wedge mapC(a,d) \wedge c \neq d) \wedge \nexists e(e \in O_{new} \wedge mapC(b,e) \wedge c \neq e) \rightarrow \textbf{create}[merge(\{a\},c), merge(\{b\},c)], \textbf{eliminate}[mapC(a,c), mapC(b,c)]$$

Die linke Seite der Regel (Vorbedingungen) prüft, ob mindestens zwei verschiedene Konzepte a und b ($a \neq b$) vorliegen, welche mit dem gleichen Zielkonzept c verbunden sind (*mapC*). Weiterhin wird sichergestellt, dass weder a noch b andere Zielkonzepte außer c besitzen (siehe Bedingungen $\nexists d(\ldots)$ und $\nexists e(\ldots)$). Sind die Vorbedingungen erfüllt, werden zwei einelementige *merge* Änderungsoperationen generiert (**create**), eine Operation fasst a in c zusammen, während die andere b in c fusioniert. Die zugehörigen Basisänderungsoperationen $mapC(a,c)$ und $mapC(b,c)$ werden aus der Menge der vorhandenen Änderungsoperationen gelöscht (**eliminate**). Die generierten einelementigen *merge* Operationen werden im weiteren Verlauf mittels Aggregationsregeln zu einem mehrelementigen *merge* zusammengefasst.

Für das Beispiel erzeugt Regel (c_7) aus den Operationen *mapC*('DVD-ROM','Other'), *mapC*('CD-RW','Other') und *mapC*('Other','Other') die folgenden *merge* Änderungen: *merge*({'DVD-ROM'},'Other'), *merge*({'CD-RW'},'Other') und *merge* ({'Other'},'Other').

5.3.3 Aggregation COG Regeln

Aggregation COG Regeln (a-COG) dienen der Bestimmung aller betroffenen Elemente innerhalb einer mengenwertigen komplexen Änderungsoperation. So können häufig mehrere einelementige oder auch mehrelementige Änderungsoperationen für eine kompaktere Repräsentation des Evolution-Mappings in einer einzigen Änderungsoperation zusammengefasst werden. Somit können dann redundante Änderungsoperationen, welche nun durch eine kompaktere Änderungsoperation bereits abgedeckt sind, gelöscht werden. a-COG Regeln arbeiten rekursiv, d. h. sie fassen inkrementell Elemente in Änderungsoperationen zusammen, solange eine Aggregation möglich ist. Im gleichen Schritt werden redundante bzw. temporäre Änderungsoperationen entfernt.

Beispielsweise sieht die a-COG Regel für *merge* wie folgt aus:

$$(a_3)\ \ c \in O_{new} \wedge A, B \subseteq O_{old} \wedge merge(A, c) \wedge merge(B, c) \wedge A \neq B \\ \rightarrow \textbf{create}[merge(A \cup B, c)], \textbf{eliminate}[merge(A, c), merge(B, c)]$$

Die Regel spezifiziert, dass zwei partielle *merge* Operationen operierend auf den Mengen A und B mit dem gleichen Zielkonzept c zusammengefasst werden können. Die neue, kompakte Änderungsoperation $merge(A \cup B,c)$ besitzt die Quelle $A \cup B$ und das Zielkonzept c. Da nun die *merge* Operation von A nach c wie auch die von B nach c in $merge(A \cup B, c)$ abgedeckt sind, können die beiden Ersteren aus dem Evolution-Mapping entfernt werden. Durch mehrfaches Anwenden der Regel innerhalb des Algorithmus (siehe 5.4) werden nach und nach Änderungsoperationen und ihre Elementmengen zusammengefasst bis keine weitere Aggregation mehr möglich ist. Im Beispiel würde im ersten Schritt die Operation *merge*({'DVD-ROM','CD-RW'},'Other') erzeugt werden, die Operationen *merge*('DVD-ROM','Other') und *merge*('CD-RW','Other') werden dabei gelöscht. Anschließend wird die endgültige *merge* Operation *merge*({'DVD-ROM','CD-RW','Other'},'Other') generiert und *merge*({'DVD-ROM','CD-RW'},'Other') sowie *merge*({'Other'},'Other') werden entfernt.

5.4 Diff Algorithmus

Dieser Abschnitt beschreibt den Ansatz zur Bestimmung von Diff Evolution-Mappings. Zu Beginn wird ein Überblick über den Ansatz gegeben und gezeigt wie das für die Berechnung benötigte Match-Mapping erstellt wird. Danach werden die Algorithmen zur Bestimmung des Basis-Evolution-Mappings $diff_{basic}$ sowie des finalen Evolution-Mappings $diff_{compact}$ vorgestellt. Zudem wird die Korrektheit der Algorithmen bewiesen und deren Komplexität diskutiert. Am Ende des Abschnitts wird auf die Verwendung des Diff Evolution-Mappings und seiner Inversen zur Migration von Ontologieversionen eingegangen.

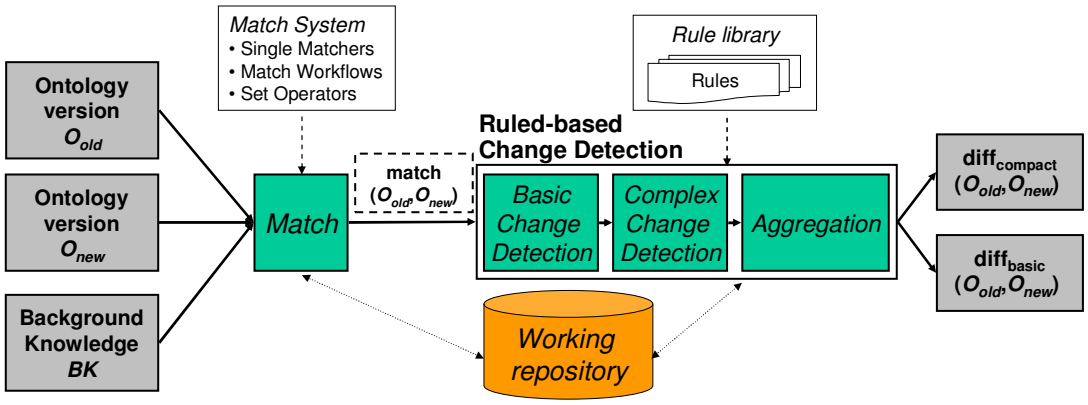


Abbildung 5.2: Schematischer Überblick über den regelbasierten Diff-Ansatz

5.4.1 Überblick

Die verschiedenen Phasen des Algorithmus sind in Abb. 5.2 dargestellt. Die Eingabe sind zwei Ontologieversionen (O_{old}, O_{new}) einer Ontologie O. Falls vorhanden kann zusätzliches Hintergrundwissen (background knowledge BK) z. B. Synonym-Verzeichnisse oder Wörterbücher, zum Abgleich der Ontologieversionen verwendet werden. Das Ergebnis des Algorithmus beinhaltet ein ausdrucksstarkes ($diff_{compact}(O_{old}, O_{new})$) sowie ein einfaches ($diff_{basic}(O_{old}, O_{new})$) Diff Evolution-Mapping. $diff_{basic}$ enthält nur einfache Änderungsoperationen und kann zur Migration von Ontologieversionen verwendet werden. Des Weiteren kann optional zu jedem Evolution-Mapping dessen Inverse bestimmt werden (siehe 5.4.6). Der gesamte Algorithmus nutzt ein Working-Repository, welches Funktionalitäten zum Verwalten der Ontologieversionen sowie temporärer Evolution-Mappings anbietet.

Die erste Phase des Algorithmus ist ein *Matching* der beiden Ontologieversionen. Dabei werden gleiche oder ähnliche Konzepte zwischen den beiden Versionen ermittelt. Das Ergebnis ist ein Match-Mapping $match(O_{old}, O_{new})$, welches eine Menge von $matchC$ Korrespondenzen enthält.

Die nachfolgende *Rule-based Change Detection* Phase basiert auf dem Matchergebnis und läuft vollständig automatisch ab. So werden die in Abschnitt 5.3 vorgestellten COG Regeln in mehreren Schritten (*Basic Change Detection* für einfache Änderungen, *Complex Change Detection* für komplexe Änderungen und *Aggregation* für aggregiert-komplexe Änderungen) zur Bestimmung der Diff Evolution-Mappings angewandt.

5.4.2 Matching Phase

Die Matching Phase benutzt sowohl Informationen aus den beiden Ontologieversionen als auch optional Hintergrundwissen, um ein Match-Mapping zwischen den Versionen herzuleiten. Das Matching von Ontologieversionen ist typischerweise leichter als die Bestimmung eines Match-Mappings zwischen zwei unabhängig entwickelten Ontologien. Dies ist begründet durch die Abhängigkeit zwischen den beiden Versionen, d. h. die eine Version geht aus der anderen aufgrund von Änderungen hervor. Ein Großteil der Ontologie bleibt dabei oftmals unverändert. Zudem kann bei Ontologien in den Lebenswissenschaften auf die eindeutigen ID-Attribute von Konzepten (accession numbers) zurückgegriffen werden, um Korrespondenzen zwischen Versionen herzuleiten. Für alle anderen Ontologien können bereits existierende Match-Systeme und Tools (z. B. COMA++ [5]), welche eine Vielzahl von Matchern und Match-Techniken zur Verbesserung der Match-Qualität anbieten, wiederverwendet werden. Da das Matching ein semi-automatischer Prozess ist, kann ein Nutzer das automatisch bestimmte Matchergebnis zudem prüfen und Korrespondenzen verwerfen bzw. korrigieren. Der weitere Diff Algorithmus erwartet als Ergebnis der Matching Phase ein korrektes und komplettes Match-Mapping.

Für das motivierende Beispiel sieht das Ergebnis der Matching Phase wie folgt aus. Alle weiß hinterlegten Kategorien mit exakt dem gleichen Namen in der alten und neuen Version bilden Korrespondenzen (siehe Abb. 5.1). Weiterhin gehören alle mit einer gestrichelten Linie verbundenen Kategorien zur Menge der Korrespondenzen des Match-Mappings.

5.4.3 Regelbasierte Änderungserkennung

Algorithmus 1 zeigt die Implementierung des Diff Algorithmus (*diffEvolMapGen*) zur Generierung des einfachen und komplexen Diff Evolution-Mapping.

Algorithmus 1: diffEvolMapGen

Input : two ontology versions O_1 and O_2, match mapping $M = match(O_1, O_2)$,
list of rules $R = [R_{b-COG}, R_{c-COG}, R_{a-COG}]$
Output : diff evolution mappings $diff_{basic}(O_1, O_2)$, $diff_{compact}(O_1, O_2)$
$diff_{basic}(O_1, O_2) \leftarrow$ `diffBasicGen(`O_1, O_2, M, R_{b-COG}`)`;
$D \leftarrow diff_{basic}(O_1, O_2)$;
foreach $r \in R_{c-COG}$ **do**
  $D \leftarrow$ `applyRule(`D, r`)`;
end
$diff_{compact}(O_1, O_2) \leftarrow$ `applyAggRules(`D, R_{a-COG}`)`;
return $[diff_{basic}(O_1, O_2), diff_{compact}(O_1, O_2)]$;

Die Eingabe sind zwei Ontologieversionen O_1 und O_2, ein Match-Mapping M zwi-

schen O_1 und O_2 sowie eine Liste von COG Regeln R. Das Ergebnis besteht aus den beiden Diff Evolution-Mappings $diff_{basic}(O_1, O_2)$ (nur einfache Änderungsoperationen) und $diff_{compact}(O_1, O_2)$ (semantisch ausdrucksstarkes Evolution-Mapping). In den drei Hauptschritten des Algorithmus werden die drei zuvor vorgestellten Typen von COG Regeln angewandt, um die entsprechenden Änderungsoperationen zu generieren.

Im ersten Schritt des Hauptalgorithmus wird eine Prozedur *diffBasicGen* aufgerufen, um auf Basis des Match-Mappings M zwischen den beiden Versionen O_1 und O_2 sowie den b-COG Regeln R_{b-COG} das Basis Evolution-Mapping $diff_{basic}(O_1, O_2)$ zu bestimmen. Der Algorithmus von *diffBasicGen* sieht wie folgt aus:

Algorithmus 2: diffBasicGen

Input : two ontology versions O_1 and O_2, match mapping $M = match(O_1, O_2)$, list of b-COG rules R_{b-COG}
Output : basic diff evolution mapping $D = diff_{basic}(O_1, O_2)$
$D \leftarrow empty$;
foreach $r \in R_{b-COG}$ **do**
  $D \leftarrow$ `applyBasicRule`(D, r, O_1, O_2, M);
end
return D;

Alle b-COG Regeln werden exakt einmal in einer vorgegebenen Reihenfolge (siehe Nummerierung im Anhang B.2) angewandt (*applyBasicRule*). *applyBasicRule* wertet alle Elemente von O_1 bzw. O_2 aus, inwieweit sie für die aktuelle Regel relevant erscheinen oder nicht. Falls ja, werden die entsprechenden Änderungsoperationen erzeugt und im Evolution-Mapping erfasst. Im späteren Abschnitt 5.4.4 wird die Korrektheit von *diffBasicGen* bewiesen. Das Ergebnis wird in 5.4.5 für die Migration von Ontologieversionen eingesetzt.

Die Verarbeitung der c-COG Regeln in *diffEvolMapGen* ist vergleichbar mit der Verarbeitung der b-COG Regeln, d. h. jede Regel wird exakt einmal in einer vorgegebenen Reihenfolge angewandt. Die Verarbeitung startet auf dem Basis-Evolution-Mapping und erweitert dieses iterativ durch komplexe Änderungsoperationen, welche über die c-COG Regeln generiert werden. Über *eliminate* Aktionen werden gleichzeitig nicht mehr benötigte Änderungsoperationen entfernt, da sie zuvor durch neu erzeugte komplexe Änderungsoperationen ersetzt wurden.

Die Aggregation erfordert die Mehrfachanwendung von a-COG Regeln, um rekursiv die mengen-basierten Änderungsoperationen zu erzeugen. Diese Funktionalität ist innerhalb des *applyAggRules* Algorithmus realisiert, welcher im letzten Schritt des Hauptalgorithmus aufgerufen wird.

Algorithmus 3: applyAggRules

Input : diff evolution mapping $D(D' \leftarrow D)$, list of a-COG rules R_{a-COG}
Output : diff evolution mapping D'
repeat
  $D \leftarrow D'$;
  foreach $r \in R_{a-COG}$ **do**
    $D' \leftarrow$ `applyRule(`D', r`)`;
  end
until $D \neq D'$;
return D';

Der Algorithmus *applyAggRules* erwartet ein Diff Evolution-Mapping D und eine geordnete Liste von a-COG Regeln R_{a-COG} als Eingabe. In jeder Runde (repeat Schleife) werden die in R_{a-COG} befindlichen Regeln in ihrer Reihenfolge angewandt (Nummerierung und Reihenfolge siehe Anhang B.2). So können a-COG Regeln mehrfach angewandt werden (einmal pro Runde), um rekursiv die mengenwertigen Änderungsoperationen zu erzeugen. Die Anwendung einer Regel (*applyRule*) modifiziert das temporäre Evolution-Mapping D' entsprechend ihrer Aktionen (create oder eliminate), falls die Vorbedingungen erfüllt sind und zu erzeugende Änderungen noch nicht vorliegen. Die Regeln werden solange angewandt bis das temporäre Evolution-Mapping sich nicht mehr ändert ($D \neq D'$).

Anhang B.3 zeigt die Ergebnisse sowie alle Zwischenresultate für die Ausführung des Algorithmus *diffEvolMapGen* auf dem motivierenden Beispiel aus Abb. 5.1. Zur Illustration wird an dieser Stelle auf die Bestimmung der komplexen Änderungsoperation „Hinzufügung des Subgraphen 'Solid State Disks'“ näher eingegangen. So erzeugt für das Subgraph Beispiel Regel (b_1) fünf relevante Hinzufügungen von Konzepten: $addC$(Solid State Disks), $addC$(SLC), $addC$(MLC), $addC$(1.3) und $addC$(0.85). Zur Erkennung von Subgraph-Einfügungen werden die beiden folgenden c-COG Regeln benötigt:

$$(c_5)\ a, r \in O_{new} \wedge addC(a) \wedge addR(r) \wedge a = r_{source} \wedge \nexists s(s \in O_{new} \wedge addR(s) \wedge a = s_{target}) \rightarrow \textbf{create}[addLeaf(a, \{r_{target}\})], \textbf{eliminate}[addC(a), addR(r)]$$

$$(c_9)\ a, b \in O_{new} \wedge B \subseteq O_{new} \wedge addC(a) \wedge addLeaf(b, B) \wedge a \in B \rightarrow \textbf{create}[addSubGraph(a, \{b\})], \textbf{eliminate}[addC(a), addLeaf(b, B)]$$

Die erste Regel (c_5) wird zur Generierung von Einfügungen von Blattkonzepten verwendet. Regel (c_9) basiert auf den Ergebnissen von (c_5) und bestimmt Subgraph-Einfügungen bestehend aus einem neu hinzugefügten Konzept a sowie einem hinzugefügten Blattkonzept b unterhalb von a. Im Beispiel werden die Konzepte '1.3' sowie '0.85' als hinzugefügte Blattkonzepte klassifiziert. Aufbauend darauf generiert Regel (c_9) die beiden komplexen Änderungen $addSubGraph$(SLC,{1.3}) und

addSubGraph(MLC,{0.85}) . Anschließend können die folgenden a-COG Regeln angewandt werden:

(a_5) $a, b, r \in O_{new} \wedge A \subseteq O_{new} \wedge addSubGraph(a, A) \wedge addC(b) \wedge addR(r) \wedge$
$r_{source} = a \wedge r_{target} = b$
$\rightarrow$ **create**$[addSubGraph(b, \{a\} \cup A)]$,
eliminate$[addSubGraph(a, A), addC(b), addR(r)]$

(a_6) $a \in O_{new} \wedge A, B \subseteq O_{new} \wedge addSubGraph(a, A) \wedge addSubGraph(a, B) \wedge A \neq B$
$\rightarrow$ **create**$[addSubGraph(a, A \cup B)]$,
eliminate$[addSubGraph(a, A), addSubGraph(a, B)]$

Regel (a_5) aggregiert rekursiv hinzugefügte Konzepte zu größeren Subgraphen. Existieren mehrere Subgraph-Einfügungen mit der gleichen Wurzel, so können diese in einer Subgraph-Einfügung kombiniert werden, dabei werden die Subkonzepte in einer Menge vereinigt (a_6). Im Beispiel würde Regel (a_5) zwei komplexe Änderungen erzeugen: *addSubGraph*(Solid State Disks, {SLC,1.3}) und *addSubGraph*(Solid State Disks, {MLC,0.85}) . Diese werden anschließend durch Regel (a_6) zu *addSubGraph*(Solid State Disks, {SLC, 1.3, MLC, 0.85}) zusammengefasst.

Das finale Ergebnis ist das semantisch reichhaltigste Evolution-Mapping, welches bzgl. der definierten Regeln nicht weiter verkleinert werden kann. So bleibt für das Subgraph Beispiel nur die komplexe Änderungsoperation *addSubGraph*(Solid State Disks, {SLC, 1.3, MLC, 0.85}) übrig und eine weitere Aggregation ist nicht möglich. Alle zwischenzeitlich erzeugten Änderungsoperationen wurden während der Regelanwendung entfernt. Wie Anhang B.3 zeigt, besteht das finale Diff Evolution-Mapping $diff_{compact}$ für das motivierende Beispiel aus lediglich sechs Änderungen, wohingegen das einfache Evolution-Mapping $diff_{basic}$ 25 Änderungen beinhaltet.

Die Komplexität des Algorithmus *diffEvolMapGen* hängt von der Anzahl der Regelanwendungen sowie der Anzahl veränderter Ontologieelemente n' ab. b-COG und c-COG Regeln werden lediglich einmal für alle veränderten Ontologieelemente angewandt. Da die Anzahl geänderter Elemente n' niemals größer sein kann als die Anzahl aller Ontologieelemente n, d. h. jedes Ontologieelement wird maximal von einer einfachen Änderung (*add*, *del*, *map*) erfasst, kann n' mit n nach oben hin abgeschätzt werden. So ergibt sich für die beiden Regeltypen eine Komplexität von $O(n)$ falls eine Ontologie mit n Ontologieelementen vorliegt. Die rekursiven a-COG Regeln reduzieren inkrementell die vorhandenen Änderungsoperationen, da bei jeder Regelanwendung mindestens zwei Änderungen zu einer zusammengefasst werden. Dies ergibt $O(log(n))$ Regelanwendungen und damit eine totale Komplexität von $O(n \log(n))$. Da typischerweise die geänderten Ontologieelemente nur einen kleinen Bruchteil der Gesamtontologie ausmachen, können schnelle Laufzeiten auch für große Ontologien erwartet werden.

5.4.4 Korrektheit

Die Korrektheit des vorgestellten Diff Algorithmus beinhaltet zwei Aspekte. Erstens muss der Algorithmus alle Änderungen finden (Vollständigkeit) und zweitens muss er terminieren, d. h. in endlicher Zeit ein Diff Evolution-Mapping berechnen. Es wird zuerst gezeigt, dass das erzeugte einfache Evolution-Mapping komplett ist, d. h. es enthält alle einfachen Änderungen zwischen zwei Ontologieversionen O_{old} und O_{new}. Es wird dabei auf Konzepte fokussiert; der Beweis für Änderungen an Beziehungen bzw. Attributen verläuft in ähnlicher Art und Weise.

Theorem 1. *Die b-COG Regeln, welche im Algorithmus diffBasicGen angewandt werden, generieren ein vollständiges Basis-Evolution-Mapping* $diff_{basic}(O_{old}, O_{new})$ *mit folgendem Inhalt:*

a) alle Konzept-Einfügungen (addC) zwischen O_{old} *und* O_{new}

b) alle Konzept-Löschungen (delC) zwischen O_{old} *und* O_{new}

c) alle Konzept-Änderungen (mapC) inklusive Konzepte, welche Korrespondenzen zu mehreren Konzepten der anderen Ontologieversion aufweisen

Der Beweis des Theorems greift die fünf in Abschnitt zuvor definierten b-COG Regeln (5.3.1) auf, welche im Algorithmus *diffBasicGen* Anwendung finden. Die fünf b-COG Regeln unterscheiden grundsätzlich zwischen Konzepten, welche mit mindestens einem Konzept in der anderen Ontologieversion matchen und jenen die keinen Matchpartner aufweisen. Für alle nicht-matchenden Konzepte in O_{new} generiert b-COG Regel (b_1) *addC* Änderungsoperationen. Umgekehrt erzeugt b-COG Regel (b_2) Löschungen (*delC*) für alle nicht-matchenden Konzepte in O_{old}. Matchende Konzepte treten in Korrespondenzen $matchC(a, b) \in match(O_{old}, O_{new})$ auf und werden durch die b-COG Regeln (b_3), (b_4) und (b_5) verarbeitet. So generiert Regel (b_3) eine $mapC(a, b)$ Änderungsoperation, falls sich beide Konzepte a und b unterscheiden ($a \neq b$). Falls ($a = b$) wird über die Regeln (b_4) und (b_5) sicher gestellt, dass nur *mapC* Änderungen für Konzepte, die mehr als eine Korrespondenz aufweisen, erzeugt werden. Da in diesen Fällen eine Änderung an den entsprechenden Konzepten vorliegt, muss diese im Diff Evolution-Mapping repräsentiert werden. Demzufolge werden $matchC(a, a)$ Korrespondenzen zwischen unveränderten Konzepten a nicht im Diff Evolution-Mapping erfasst. Zusammenfasst enthält $diff_{basic}$ alle einfachen Änderungen und blendet die unveränderten Ontologieteile aus.

Um die Vollständigkeit von $diff_{compact}$ zu beweisen, muss gezeigt werden, dass der Algorithmus *diffEvolMapGen* das semantisch reichhaltigste Diff Evolution-Mapping bzgl. der vorgegebenen c-COG und a-COG Regeln generiert. Die Vollständigkeit

wird dabei durch drei Fakten garantiert. Erstens dient das Basis-Evolution-Mapping $diff_{basic}$ als Eingabe zur Berechnung von $diff_{compact}$. Dessen Vollständigkeit wurde bereits gezeigt (siehe Theorem 1). Zweitens decken die Regeln alle vorgestellten Änderungsoperationen ab und werden iterativ für alle möglichen Eingabekonfigurationen angewandt solange neue Änderungen abgeleitet werden können. Drittens terminiert der Algorithmus, was im Folgenden gezeigt werden soll.

Die Terminierung des Algorithmus *diffEvolMapGen* ist hauptsächlich in zwei Fakten begründet. Alle Regeln arbeiten auf einer endlichen Menge von Ontologieelementen aus O_{old} bzw. O_{new}. Zudem werden durch die Regeln keine neuen Ontologieelemente erzeugt und existierende Elemente werden lediglich in Änderungsoperationen verarbeitet. Zweitens terminieren alle Regeln, da diese entweder einmalig angewandt werden oder solange sich das Evolution-Mapping ändert. So werden b-COG und c-COG Regeln lediglich einmal, also nicht-rekursiv, in einer vorgegebenen Reihenfolge angewandt. a-COG Regeln sind zwar rekursiv, d. h. sie werden unter Umständen mehrmals angewandt, besitzen jedoch eine entscheidende Eigenschaft. Bei jeder Anwendung einer a-COG Regel reduziert sich die Anzahl vorhandener Änderungsoperationen indem Ontologieelemente zusammengefasst werden. So werden in jeder a-COG Regel mindestens zwei Änderungsoperationen in einer zusammengefasst. Diese stetige Reduzierung der Änderungsoperationen führt dazu, dass eine Terminierung einsetzt, wenn die kompaktesten Änderungsoperationen ermittelt wurden.

Bei Ergänzungen der Änderungsoperationen bzw. Regelmenge, beispielsweise durch geänderte Anforderungen in einer Anwendung, wird die Korrektheit des Algorithmus durch Prüfung der Charakteristika der COG Regeln erhalten. So dürfen keine zyklischen Abhängigkeiten zwischen Regeln entstehen und rekursive a-COG Regeln müssen stets die Anzahl von Änderungsoperationen reduzieren.

5.4.5 Migration von Ontologieversionen

Eine wichtige Anwendung des Diff Evolution-Mappings ist die Migration von Ontologieversionen. So kann mit Hilfe eines Basis-Evolution-Mappings $diff_{basic}(O_1, O_2)$ die alte Ontologieversion O_1 in O_2 überführt (migriert) werden. Diese Art der Migration verfolgt eine „in-place“ Ersetzung (Anpassung) von Ontologieelementen. Dabei bleiben unveränderte Ontologieelemente unberührt. Somit sind Änderungen auf die von Evolution betroffenen Ontologieteile limitiert, was zu einer effektiven Migration der Ontologieversion führt. Im Gegensatz dazu würde das Erstellen einer komplett neuen Ontologieversion alle Ontologieelemente betreffen.

Der Algorithmus *ontVersionMig* zeigt die Implementierung der Migration für eine Ontologieversion O_1 nach O_1' unter Nutzung des Evolution-Mappings $diff_{O_1,O_2}$.

Algorithmus 4: ontVersionMig

Input : ontology version O_1, basic diff evolution mapping $D = diff_{basic}(O_1, O_2)$
Output : migrated ontology version O_1'
$O_1' \leftarrow O_1$;
$performOrder \leftarrow$ [delA,delR,delC,mapC,mapA,mapR,addC,addA,addR];
foreach $chgOp \in performOrder$ **do**
  $O_1' \leftarrow$ `perform`$(D.getChgOp(chgOp), O_1')$;
end
return O_1';

Der Migrationsalgorithmus wendet die einfachen Änderungsoperationen aus $diff_{basic}$ (O_1, O_2) in einer vorgegebenen Reihenfolge $performOrder$ an. Für Löschungen werden zunächst die Attribute entfernt ($delA$). Danach werden die Konzepte aus der Ontologiestruktur entfernt ($delR$). Abschließend werden die Konzepte selbst gelöscht ($delC$). Bei map Änderungen werden zunächst Konzepte ersetzt ($mapC$), danach können $mapR$ und $mapA$ Änderungen ausgeführt werden, z. B. das Ersetzen eines Attributwertes oder des Typs einer Beziehung. Einfügungen werden am Ende ausgeführt. Dabei werden zunächst werden die neuen Konzepte hinzugefügt ($addC$) danach ihre Attribute ($addA$) und Beziehungen ($addR$).

Theorem 2 beschreibt die Korrektheit des Migrationsalgorithmus:

Theorem 2. *Der Algorithmus ontVersionMig arbeitet korrekt, d. h. für ein Basis-Evolution-Mapping $diff_{basic}(O_1, O_2)$, das durch den Algorithmus diffBasicGen bestimmt wurde, kann ontVersionMig die Ontologieversion O_2 aus O_1 erzeugen.*

Der nachfolgende Beweis für Theorem 2 beschränkt sich auf Änderungen an Konzepten. Für Änderungen an Attributen und Beziehungen kann eine analoge Beweisführung durchgeführt werden. Es muss gezeigt werden, dass die erzeugte Ontologieversion vollständig ist, d. h. sie enthält alle Konzepte und keine weiteren oder anderen Konzepte. Es ist erstens relativ einfach zu erkennen, dass der Algorithmus gelöschte Konzepte entfernt (siehe $delC$ Änderungen), so dass diese Konzepte nicht in O_2 vertreten sind. In gleicher Art und Weise werden Konzepte in $mapC$ Änderungen entfernt, da sie ersetzt werden und somit ebenfalls nicht in O_2 auftreten. Zweitens werden unveränderte Konzepte nicht durch Änderungsoperationen im Evolution-Mapping erfasst, d. h. ein unverändertes Konzept c aus O_1 wird ebenfalls in O_2 vorkommen. Drittens werden neue Konzepte durch $addC$ Änderungen in den unveränderten Teil der Ontologie eingefügt und sind somit Teil der neuen Version O_2. Analog werden die Zielkonzepte von $mapC$ Änderungen aus $diff_{basic}$ in die Ontologie eingefügt.

5.4.6 Inverse Evolution-Mappings

Inverse Diff Evolution-Mappings können angewendet werden, um Änderungen rückgängig zu machen („undo-Funktionalität"). Aus einer geänderten Ontologieversion kann somit die ursprüngliche Version vor den Änderungen wiederhergestellt werden. Das vorgestellte Änderungsmodell, bestehend aus Evolution-Mappings und Änderungsoperationen, erlaubt eine einfache und effektive Berechnung inverser Evolution-Mappings. Da jede Änderungsoperation exakt einer inversen Änderungsoperation zugeordnet werden kann (siehe 5.2 und Anhang B.1), ist es möglich ein komplettes Evolution-Mapping, bestehend aus einer Menge von Änderungsoperationen, zu invertieren, d. h. das inverse Evolution-Mapping zu erzeugen. So besteht das inverse Evolution-Mapping aus der Menge der invertierten Änderungsoperationen des eigentlichen Evolution-Mapping.

Theorem 3*. Das Inverse eines Basis-Evolution-Mappings* $diff_{basic}(O_1, O_2)$ *ist korrekt, d. h. es ist identisch zu* $diff_{basic}(O_2, O_1)$*.*

Der Beweis des Theorems konzentriert sich auf Änderungen an Konzepten. Die Beweisführung für Änderungen an Attributen und Beziehungen verläuft analog. Nachfolgend wird gezeigt, dass die Inverse einer jeden Änderungsoperation aus $diff_{basic}(O_1, O_2)$ ebenfalls in $diff_{basic}(O_2, O_1)$ vorliegt und dass $diff_{basic}(O_2, O_1)$ keine weiteren Änderungsoperationen aufweist. Die Einfügung eines neuen Konzepts c ($addC(c)$) in $diff_{basic}(O_1, O_2)$ besitzt $delC(c)$ als Inverse. Da c nicht in O_1 vorkommt jedoch in O_2 vorhanden ist, generiert Regel (b_2) eine Löschung in $diff_{basic}(O_2, O_1)$. Analog erzeugt die b-COG Regel (b_1) ein $addC(c)$ in $diff_{basic}(O_2, O_1)$ falls c in O_1 aber nicht in O_2 vorkommt. Dies würde einer Löschung $delC(c)$ in $diff_{basic}(O_1, O_2)$ entsprechen was gleichzeitig die Inverse von $addC(c)$ darstellt. Eine Änderung $mapC(a, b)$ in $diff_{basic}(O_1, O_2)$ besitzt als Inverse $mapC(b, a)$ und benötigt aufgrund der b-COG Regeln (b_3), (b_4) oder (b_5) eine Korrespondenz $matchC(a, b)$ im Match-Mapping zwischen O_1 und O_2. Bei Umkehr der Richtung des Match-Mapping ergibt dies eine $matchC(b, a)$ Korrespondenz und folglich aufgrund der b-COG Regeln eine $mapC(b, a)$ Änderung in $diff_{basic}(O_2, O_1)$. Die Änderungsoperationen in $diff_{basic}(O_2, O_1)$ werden wie die Änderungen in $diff_{basic}(O_1, O_2)$ ausschließlich durch die b-COG Regeln (b_1) bis (b_5) generiert. Somit können keine zusätzlichen Änderungsoperationen als die inversen Änderungen von $diff_{basic}(O_1, O_2)$ vorliegen.

Die Inverse von $diff_{basic}(O_1, O_2)$ erzeugt ein Evolution-Mapping, welches unter Nutzung des Algorithmus *ontVersionMig* zur korrekten Migration der Ontologieversion O_2 nach O_1 verwendet werden kann.

Zur Evaluierung und Verifikation der vorgeschlagenen Migrationsalgorithmen wird nachfolgend ein Migrationsszenario für zwei Ontologieversionen beschrieben. Wie in Abb. 5.3 illustriert, wird zunächst die Ontologieversion O_1 in die geänderte Version O_1' migriert: $O_1' = ontVersionMig(O_1, diff_{basic}(O_1, O_2))$. Aufgrund der Kor-

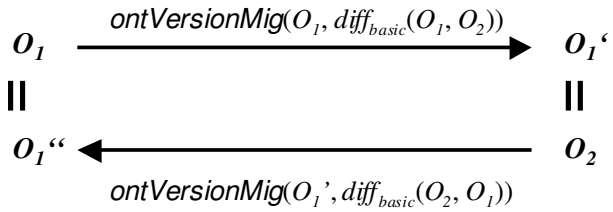


Abbildung 5.3: Migrationsszenario für Ontologieversionen

rektheit des Algorithmus *ontVersionMig* (Theorem 2) sind die Versionen O'_1 und O_2 identisch. Zweitens wird das inverse Evolution-Mapping von $diff_{basic}(O_1, O_2)$ also $diff_{basic}(O_2, O_1)$ bestimmt und zur Migration von O'_1 verwendet: $O''_1 = ontVersionMig(O'_1, diff_{basic}\ (O_2, O_1))$. Die resultierende Ontologieversion O''_1 wird mit O_1 übereinstimmen, da der Migrationsalgorithmus (Theorem 2) und das Inverse eines Evolution-Mappings (Theorem 3) korrekt sind. Das vorgeschlagene Migrationsszenario wurde ebenfalls praktisch anhand einer Reihe von Ontologien aus der Realwelt evaluiert. Zugehörige Ergebnisse werden im nachfolgenden Evaluierungsabschnitt diskutiert.

5.5 Evaluierung

Die Evaluierung des Ansatzes umfasst die Bestimmung von Evolution-Mappings für zwei Realwelt-Ontologien: Gene Ontology (GO) [36] sowie einem Ausschnitt des DMoz Webverzeichnisses[22] für Fußball (Soccer). GO ist eine der weitverbreitesten Ontologien in der Bioinformatik und dient der einheitlichen Annotation von molekular-biologischen Objekten (z. B. Proteinen). Das DMoz Webverzeichnis stellt derzeit das größte manuell erstellte Webverzeichnis dar und wird in vielen Web-Portalen und Suchmaschinen wie Google oder Lycos verwendet. Für die Evaluierung wurden zwei Jahresperioden für beide Ontologien untersucht: 2008 (2008-01 → 2009-01) und 2009 (2009-01 → 2010-01). Das Matching der Ontologieversionen fand auf unterschiedliche Art und Weise statt. Für GO Versionen wurden die vorhandenen eindeutigen Identifier der Konzepte verwendet, um das Match-Mapping zu erzeugen. Da es in DMoz keine Identifier gibt, wurde ein semi-automatisches Match-Verfahren eingesetzt, welches Informationen aus den Labels der Kategorien sowie der Struktur nutzte. GO beinhaltet Konzepte, Beziehungen (is_a, part_of) wie auch Attribute. DMoz hingegen ist eine reine is_a Hierarchie und enthält neben dem Label einer Kategorie keine weiteren Attribute. Um die Vergleichbarkeit zu ermöglichen, werden in den nachfolgenden Analysen lediglich Änderungen an Konzepten und Beziehungen berücksichtigt. Die Implementierung des vorgestellten Ansatzes erfolgte in Java unter Verwendung einer MySQL Datenbank als Repository.

[22] http://www.dmoz.org

O_{old} - O_{new}	$\|O_{old}\|$	$\|O_{new}\|$	|match|	$\|diff_{compact}\|$
$GO_{2008\text{-}01}$ - $GO_{2009\text{-}01}$	66,121	75,180	25,774	8,450
$GO_{2009\text{-}01}$ - $GO_{2010\text{-}01}$	75,180	84,714	27,861	4,284
$DMoz_{2008\text{-}01}$ - $DMoz_{2009\text{-}01}$	6,683	6,185	3,130	255
$DMoz_{2009\text{-}01}$ - $DMoz_{2010\text{-}01}$	6,185	6,079	3,047	65

Tabelle 5.1: Übersichtsstatistiken für GO und DMoz

Tab. 5.1 zeigt die Größen der Ontologien bzw. Mappings für die untersuchten Zeiträume. GO ist bedeutend größer als DMoz und erfuhr in den beiden Perioden einen enormen Zuwachs. DMoz verkleinerte sich hingegen, insbesondere in der ersten Periode (2008). Die beiden letzten Spalten stellen die Größen von *match* und $diff_{compact}$ dar. Die Ergebnisse zeigen, dass der Diff Algorithmus im Vergleich zur Größe der jeweils neuen Ontologieversion ein relativ kompaktes Diff Evolution-Mapping berechnet. So ist $diff_{compact}(GO_{2009\_01}, GO_{2010\_01})$ um Faktor 20 kleiner als die vollständige neue Version GO_{2010_01}. Im Fall von DMoz ist die Kompaktheit noch deutlicher zu erkennen. Beispielsweise besitzt $diff_{compact}(DMoz_{2009\_01}, DMoz_{2010\_01})$ eine um Faktor 93 geringere Größe. Dies zeigt, dass der Algorithmus in der Lage ist kompakte Diff Evolution-Mappings zu bestimmen, welche lediglich die Änderungen zwischen den Versionen enthalten. Somit könnten beispielsweise Anbieter von Ontologien neben den neuen Versionen ebenfalls ein kompaktes Evolution-Mapping bereitstellen, um Nutzern die Transformation oder Migration ihrer Daten und Anwendungen zu erleichtern. Außerdem würde nur eine Veröffentlichung des Evolution-Mapping ausreichen, da Anwender dann automatisch ihre derzeitige Version ohne ein komplettes Laden der neuen Version aktualisieren könnten.

Die Berechnung der Diff Evolution-Mappings benötige ca. 150 s für GO und lediglich 5 s für DMoz (jeweils bezogen auf 2009). Dabei waren für GO 9 und für DMoz 3 Iterationen während der Regelanwendung nötig. Dies liegt zum einen an der geringeren Größe von DMoz und zum anderen am geringen Auftreten komplexer Änderungsoperationen wie das Einfügen großer Subgraphen.

Die berechneten Evolution-Mappings wurden im Detail noch weiter begutachtet. Tab. 5.2 illustriert dazu die absolute Häufigkeit komplexer und einfacher Änderungsoperationen. Zwischen beiden Ontologien bestehen signifikante Unterschiede. Der enorme Zuwachs von GO spiegelt sich in einer großen Anzahl informationserweiternder Änderungen wie *addLeaf* oder der hohen Anzahl von Subgraph-Einfügungen wieder. So umfasst beispielsweise der größte hinzugefügte Subgraph (GO:0070887, 'cellular response to chemical stimulus') 94 neue Konzepte. In GO wurden keine Konzepte gelöscht, da nicht mehr benötige Konzepte weiterhin unter dem Status veraltet (obsolete) in der Ontologie geführt werden. Somit verbleibt veraltetes Wis-

	GO		DMoz	
	2008	**2009**	**2008**	**2009**
add	4,187	1,355	0	0
del	1,407	316	0	0
map	0	0	0	0
addLeaf	768	796	18	6
delLeaf	0	0	199	46
merge	70	83	16	4
move	1,499	1,200	0	0
substitute	0	1	15	9
toObsolete	225	66	0	0
addSubGraph	294	467	0	0
delSubGraph	0	0	7	0
∑	**8,450**	**4,284**	**255**	**65**

Tabelle 5.2: Verteilung der Änderungsoperationen in $diff_{compact}$

sen aus Kompatibilitätsgründen in der Ontologie erhalten. Im Gegensatz zu GO liegen in DMoz neben wenigen Einfügungen vorwiegend Löschungen vor. Dies zeigt, dass dieser Teil des Webverzeichnisses überarbeitet und dabei hauptsächlich konsolidiert wurde. Anzumerken ist, dass die finalen Evolution-Mappings für DMoz keine einfachen Änderungsoperationen enthalten, d. h. alle einfachen Änderungen konnten durch komplexe Änderungen abgedeckt werden.

Des Weiteren wurden die Größen der einfachen $diff_{basic}$ sowie des ausdrucksstarken Diff Evolution-Mappings $diff_{compact}$ analysiert und miteinander verglichen. Dazu wurde zusätzlich der Anteil einfacher und komplexer Änderungsoperationen in $diff_{compact}$ bestimmt. Tab. 5.3 zeigt ebenfalls das Verhältnis beider Größen $\frac{|diff_{compact}|}{|diff_{basic}|}$, welches zwischen 31 und 54% liegt an. Für DMoz konnten alle einfachen Änderungen durch komplexe Änderungen im finalen Evolution-Mapping ersetzt wer-

	GO		DMoz	
	2008	**2009**	**2008**	**2009**
|diff $_{basic}$|	15,781	13,504	630	149
|diff$_{compact}$|	8,450	4,284	255	65
#basic	5,594	1,671	0	0
#complex	2,856	2,613	255	65
ratio in %	**53.5%**	**31.7%**	**40.5%**	**43.6%**

Tabelle 5.3: Vergleich von $diff_{basic}$ und $diff_{compact}$

$diff(O_1,O_2)$	$\|O_1\|$	$\|diff_{basic}\|$	$\|O_1 \cap O_1''\|$	$\|O_1 \cup O_1''\|$
$GO_{2008\text{-}01}$ - $GO_{2009\text{-}01}$	200,169	29,944	200,169	200,169
$GO_{2009\text{-}01}$ - $GO_{2010\text{-}01}$	218,176	33,555	218,176	218,176
$DMoz_{2008\text{-}01}$ - $DMoz_{2009\text{-}01}$	6,683	630	6,683	6,683
$DMoz_{2009\text{-}01}$ - $DMoz_{2010\text{-}01}$	6,185	149	6,185	6,185

Tabelle 5.4: Ergebnisse der Ontologiemigration für GO und DMoz

den. Für GO waren in 2009 die Anzahl komplexer Änderungen nahezu doppelt so groß wie die noch verbleibenden einfachen Änderungen.

In der letzten Analyse wurde die in 5.4.6 vorgeschlagene Migration von Ontologieversionen für beide Ontologien praktisch evaluiert. Um einen Vergleich der Ontologieversionen O_1 und O_1' zu ermöglichen, wurde eine Repräsentation der Ontologie über Mengen verwendet. Es werden Mengen von Konzepten, Beziehungen und Attributen unterschieden. Durch Auswertung von $O_1 \cap O_1'' = O_1 \cup O_1''$ wird geprüft, ob die migrierte Version O_1'' exakt die gleichen Ontologieelemente (Konzepte, Beziehungen, Attribute) wie O_1 aufweist. Ist dies der Fall, liegt ein vollständiges Evolution-Mapping, eine vollständige Inverse sowie eine korrekt durchgeführte Migration vor. Die Ergebnisse der durchgeführten Migrationen sind in Tab. 5.4 dargestellt. Die ersten beiden Spalten der Tabelle zeigen die Anzahl der Ontologieelemente in der Ausgangsversion ($|O_1|$) sowie im Evolution-Mapping ($|diff_{basic}(O_1, O_2)|$). Erfasst werden dabei Konzepte, Beziehungen und Attribute. Die beiden letzten Spalten stellen die Anzahl der Elemente im Durchschnitt sowie in der Vereinigung der Elementmengen von O_1 und O_1'' dar. Durchschnitt und Vereinigung von O_1 und O_1'' wurden experimentell berechnet. Die resultierenden Mengen wurden miteinander verglichen und evaluiert. Es konnte nachgewiesen werden, dass O_1 und O_1'' in jedem der durchgeführten Szenarien die gleichen Ontologieelemente aufweisen. Dies zeigt, dass der in 5.4.5 vorgestellte Migrationsalgorithmus korrekt arbeitet und die berechneten Evolution-Mappings sowie ihre Inversen vollständig sind.

5.6 Zusammenfassung

In diesem Kapitel wurde ein neuartiger auf Regeln basierter Ansatz zur Bestimmung ausdrucksstarker und invertierbarer Diff Evolution-Mappings zwischen zwei Ontologieversionen vorgestellt. Ein Evolution-Mapping deckt dabei sowohl einfache als auch komplexe Änderungen ab. Der beschriebene Ansatz basiert auf einem Match-Mapping zwischen den Ontologieversionen und verwendet Change Operation Generating Rules (COG Regeln) zur Erkennung einfacher und komplexer Änderungsoperationen. Dabei geben die COG Regeln vor, wie aus einfachen Änderungsoperationen komplexe Änderungen bestimmt werden können, um ein ausdrucksstar-

kes und kompaktes Evolution-Mapping zu erzeugen. Die Evaluierung des Ansatzes erfolgte an zwei Realwelt-Ontologien aus den Lebenswissenschaften bzw. dem Web. Es konnte gezeigt werden, dass für beide Ontologien semantisch ausdrucksstarke und kompakte Evolution-Mappings generiert wurden. Des Weiteren wurden die berechneten Evolution-Mappings für die Migration von Ontologieversionen verwendet. Dabei konnte theoretisch wie praktisch gezeigt werden, dass vollständige Evolution-Mappings sowie deren Inverse eine korrekte Migration von Ontologieversionen ermöglichen. Der dargestellte Ansatz ist flexibel anpassbar und kann somit den Anforderungen verschiedener Anwendungen gerecht werden. So kann die Menge relevanter Änderungsoperationen verändert und ergänzt werden. Zudem können die zugehörigen COG Regeln an die Gegebenheiten der verschiedenen Ontologien angepasst werden.

6 Bestimmung änderungsintensiver Ontologieregionen

6.1 Motivation

Änderungen an Ontologien und das Erstellen neuer Ontologieversionen haben direkte Konsequenzen für die Nutzer bzw. deren Applikationen, welche eine Ontologie in ihren Analysen einsetzen. Üblicherweise müssen Anwender von Ontologien entscheiden, ob und wie sie eine neue Ontologieversion in ihre Anwendungen integrieren. Beispielsweise müssen Annotationen, welche auf einer geänderten Ontologie basieren, auf die neue Version angepasst werden. In diesem Prozess und insbesondere bei großen Ontologien wie in den Lebenswissenschaften erscheint es sinnvoll im Vorfeld zu wissen, welche Teile einer Ontologie besonders starken Änderungen unterlagen oder kaum Änderungen aufweisen. Mit diesem Wissen könnte u. a. der Aufwand für eine komplette Neuberechnung (z. B. Durchlauf eines komplexen Algorithmus) eingespart werden. So sollten einerseits Analysen wie z. B. Ontologie-basierte Genanalysen [16, 102] wiederholt werden, falls diese einen stark geänderten (instabilen) Ontologieteil verwenden. Die finalen Ergebnisse der Analysen sind wahrscheinlich sehr stark von den Ontologieänderungen beeinflusst und könnten sich dadurch ebenfalls verändern. Andererseits kann das Wissen über unveränderte (stabile) Ontologieteile ausgenutzt werden, um eine effiziente Neuberechnung von Ergebnissen zu ermöglichen. Beispielsweise kann beim Matching neuer Ontologieversionen für unveränderte Teile ein bereits berechnetes Ontologie-Mapping basierend auf den alten Ontologieversionen wiederverwendet werden. Dies würde eine erneute aufwendige Berechnung

des Ontologie-Mapping ersparen. Neben dem Nutzen für Applikationen und Algorithmen kann das Wissen über stabile oder instabile Ontologieteile dazu verwendet werden Designentscheidungen im Rahmen des Entwicklungsprozesses einer Ontologie zu treffen. Beispielsweise könnte entschieden werden, in welchen Teilen künftig mehr oder weniger Entwicklungsaufwand betrieben werden sollte. Somit können sich Koordinatoren von Ontologieprojekten einen Überblick über den Fortgang der kollaborativen Entwicklungsarbeiten verschaffen und künftige Arbeiten (Änderungen) an der Ontologie planen. Aufgrund der Größe von Ontologien (>10.000 Konzepte z. B. in Gene Ontology oder NCI Thesaurus) ist eine manuelle Berechnung stabiler bzw. instabiler Ontologieteile unmöglich. Es werden somit automatische Verfahren benötigt.

Dieses Kapitel präsentiert neuartiges ein Verfahren zur automatisierten Erkennung änderungsintensiver (instabiler) sowie unveränderter (stabiler) Ontologieregionen. Hierzu werden Änderungen zwischen Ontologieversionen untersucht und mit Hilfe der Ontologiestruktur entsprechende Ontologieregionen auf der Basis verschiedener Metriken selektiert. Die wesentlichen Beiträge sind:

- Es werden der Begriff Ontologieregion eingeführt und zugehörige Metriken zur Beurteilung der Änderungsintensität einer Ontologieregion definiert.
- Im Kern des Kapitels wird ein Algorithmus zur automatischen Erkennung stabiler bzw. instabiler Ontologieregionen vorgestellt. Der Algorithmus ist adaptierbar und kann somit an die Anforderungen diverser Applikationen angepasst werden. So werden erstens verschiedene Änderungstypen unterstützt. Zweitens sind die Metriken zur Bestimmung der Änderungsintensität einer Ontologieregion erweiterbar und flexibel kombinierbar. Drittens ermöglicht der Algorithmus die Erkennung von Ontologieregionen für verschiedene Zeiträume (Perioden). Somit können diverse Applikationsszenarien unterstützt werden, wie z. B. die Erkennung kleiner und instabiler Ontologieregionen oder auch großer, stabiler Ontologieregionen.
- Das vorgestellte Verfahren wurde anhand zweier großer Ontologien aus den Lebenswissenschaften evaluiert: Gene Ontology und NCI Thesaurus. Die Evaluierungsergebnisse zeigen, dass für beide Ontologien instabile sowie stabile Ontologieregionen ermittelt werden konnten. Dies zeigt, dass der vorgeschlagene Algorithmus eine automatische Erkennung änderungsintensiver Ontologieregionen in großen Ontologien ermöglicht.

6.2 Modelle und Metriken

In diesem Abschnitt werden die Grundlagen und Definitionen für den nachfolgenden Algorithmus erläutert. Für das verwendete Ontologiemodell sowie die Versio-

nierung wird auf das Grundlagenkapitel (Kapitel 3) verwiesen. Es erfolgt zunächst die Darstellung der Änderungsarten sowie die Einführung eines Kostenmodells für Änderungen. Danach wird der Begriff Ontologieregion eingeführt und zugehörige Metriken zur Beurteilung der Änderungsintensität beschrieben.

6.2.1 Änderungsarten und Kostenmodell

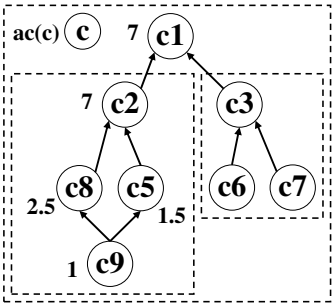
Die Evolution einer Ontologie von einer alten Version O_{old} zu einer neuen Version O_{new} kann mittels einer Menge von Änderungsoperationen beschrieben werden (siehe ebenfalls Kapitel 5 zu Evolution-Mappings). Im nachfolgenden werden die drei Änderungsarten Hinzufügung (*add*), Löschung (*del*) sowie Update (*upd*) für Konzepte, Beziehungen und Attribute unterschieden:

concept		**relationship**		**attribute**		
add	*del*	*add*	*del*	*add*	*del*	*upd*

Konzepte, Beziehungen und Attribute können eingefügt oder gelöscht werden. Für Attribute ist zusätzlich ein Update möglich, d. h. Attributwerte wie beispielsweise der Name eines Konzepts können verändert werden. Das hier verwendete Änderungsmodell enthält in der jetzigen Phase nur einfache *add*, *del* bzw. *upd* Änderungen, welche jedoch zusammengesetzt komplexe Änderungen (z. B. merge, split, ...) ergeben (siehe Kapitel 5 zu Diff). Natürlich können komplexere Änderungsoperationen später noch ergänzt werden, um eine feinere und semantisch reichhaltigere Unterscheidung zwischen auftretenden Änderungen zu erzielen.

Um den Einfluss von Änderungen auf die Konzepte einer Ontologie bewerten zu können, wird ein Kostenmodell für Änderungen eingeführt. Das Kostenmodell ordnet jeder Änderungsart Änderungskosten (numerischer Wert > 0) zu, welche den Grad des Einflusses auf die Ontologie repräsentieren. So können beispielsweise Konzept-Löschungen höhere Änderungskosten (z. B. 2 für *delConcept*) im Vergleich zu Konzept-Einfügungen (z. B. 1 für *addConcept*) zugewiesen werden.

Auf Basis des Kostenmodells für Änderungen können nun dem Konzepten, welche durch eine Änderung betroffen sind, entsprechende Änderungskosten zugewiesen werden. Hierbei wird zwischen *lokalen* (*lc*) und *aggregierten* (*ac*) Kosten für Konzepte unterschieden. Lokale Kosten $lc(c)$ eines Konzepts c decken alle Änderungen ab, welche einen direkten Einfluss auf c besitzen, d. h. Änderungen an einem Konzept selbst sowie Änderungen an dessen Attributen und Beziehungen. Beispielsweise haben Änderungen an den Kindern oder Modifikationen von Attributwerten einen direkten Einfluss auf das entsprechende Konzept und werden daher in dessen lokalen Kosten erfasst. Im späteren Algorithmus wird genauer erläutert, wie lokale Kosten von Konzepten aufgrund von erkannten Änderungen und dem Kostenmodell berechnet werden. Des Weiteren werden aggregierte Kosten $ac(c)$ verwendet, um den



OR	*abs_size*	*rel_size*	*abs_costs*	*avg_costs*
c1	8	8/8=1	7	7/8=0.875
c2	4	4/8=0.5	7	7/4=1.75
c3	3	3/8=0.375	0	0/3=0

Abbildung 6.1: Beispielontologie mit Regionen (links) und Ergebnisse der Metriken (rechts)

Einfluss von Änderungen in allen is_a Nachfolgern eines Konzepts c zu bestimmen. Somit haben Einfügungen oder Löschungen von Konzepten auch einen indirekten Einfluss auf alle Vorgängerkonzepte innerhalb der Ontologie. Die in Abb. 6.1 (links) dargestellte Beispielontologie enthält aggregierte Kosten für Konzepte (Zahlen neben den Konzepten; für Konzepte ohne dargestellte aggregierte Kosten gilt: $ac(c) = 0$). So besitzt beispielsweise Konzept $c2$ aggregierte Kosten von 7, wohingegen sein Geschwisterkonzept $c3$ aggregierte Kosten von 0 aufweist. Im späteren Algorithmus (siehe 6.3) wird gezeigt, wie aggregierte Kosten auf Basis lokaler Kosten berechnet werden können.

6.2.2 Ontologieregionen und Metriken

Eine Ontologieregion OR ist ein Subgraph innerhalb einer Ontologie, welcher ein eindeutiges Wurzelkonzept rc besitzt. Die Region OR enthält neben rc alle Konzepte, welche von rc aus mit Hilfe von is_a Beziehungen erreichbar sind (is_a Subgraph von rc). Oder anders ausgedrückt: es existiert für jedes Konzept $c \in OR$ ein is_a Pfad zu dessen Wurzel rc. Die hier eingeführte Notation einer Ontologieregion ist kompakt und basiert auf der Tatsache, dass Änderungen an Ontologien häufig im Randbereich stattfinden, beispielsweise die Einfügung von Blattkonzepten oder ganzen Subgraphen. Natürlich werden durch die Notation ebenfalls Änderungen an inneren Konzepten abgedeckt, da diese in entsprechenden Regionen innerer Konzepte erfasst werden. Als Beispiel sind in Abb. 6.1 (links) drei Ontologieregionen innerhalb der Beispielontologie markiert. So besteht u. a. die Region $c2$ aus den vier Konzepten $c2$, $c5$, $c8$ und $c9$. Die gesamte Beispielontologie mit dem Wurzelkonzept $c1$ bildet ebenfalls eine Region.

Die Änderungsintensität sowie andere Charakteristika einer Ontologieregion OR werden mit Hilfe von Metriken bestimmt. Diese Metriken können verschiedene Aspekte wie beispielsweise lokale/aggregierte Kosten, die Größe einer Region usw. beinhalten. Auf Basis der Metriken können später innerhalb des Algorithmus spezifi-

sche Ontologieregionen selektiert und erkannt werden. Es werden die nachfolgenden Metriken für Ontologieregionen verwendet:

- absolute Größe $abs\_size(OR)$: Anzahl der Konzepte innerhalb der Region OR
- relative Größe $rel\_size(OR)$: relative Größe von OR im Vgl. zur Größe der Gesamtontologie O bestimmt durch $\frac{abs\_size(OR)}{abs\_size(O)}$
- absolute Änderungskosten $abs\_costs(OR)$: absolute Änderungskosten von OR repräsentiert durch die aggregierten Kosten der Wurzel rc, d. h. $abs\_costs(OR) = ac(rc)$
- durchschnittliche Änderungskosten $avg\_costs(OR)$: die durchschnittlichen Änderungskosten pro Konzept in OR berechnet durch $\frac{abs\_costs(OR)}{abs\_size(OR)}$

Die hier gezeigten Metriken stellen eine Auswahl möglicher Metriken zur Beurteilung und Charakterisierung der Änderungsintensität von Ontologieregionen dar. Die Metriken können problemlos erweitert werden, um zusätzliche Aspekte wie beispielsweise die Tiefe einer Region oder deren Kompaktheit zu messen. Bei Anwendung der definierten Metriken auf die Regionen der Beispielontologie in Abb. 6.1 ergeben sich unterschiedliche Ergebnisse. Zwar besitzen beispielsweise $c2$ und $c3$ annähernd die gleiche Größe, unterscheiden sich jedoch stark in ihrer Änderungsintensität (abs_costs, avg_costs). Während $c3$ keine Änderungen aufweist ($avg\_costs(c3) = 0$), besitzt Ontologieregion $c2$ durchschnittliche Änderungskosten von 1,75.

6.3 Algorithmus

In diesem Abschnitt wird der Algorithmus zur automatischen Erkennung (in)stabiler Ontologieregionen erläutert. Zunächst wird gezeigt, wie aggregierte Kosten für zwei Ontologieversionen berechnet werden. Anschließend wird die Anwendung der definierten Metriken zur Bestimmung der Regionen erläutert. Aufbauend darauf wird der allgemeine Algorithmus sowie dessen Anwendbarkeit für mehrere Ontologieversionen präsentiert.

6.3.1 Aggregierte Kosten für zwei Versionen

Der Algorithmus *computeAggregatedCosts* zur Berechnung der aggregierten Kosten für zwei Versionen O_{old} und O_{new} auf Basis eines Kostenmodells σ sieht wie folgt

aus:

Algorithmus 5: computeAggregatedCosts

Input : ontology versions O_{old} and O_{new}, change costs σ
Output : ontology version O_{new} with assigned aggregated costs
$\Delta O_{old} - O_{new} \leftarrow$ `diff(`O_{old}, O_{new}`)`;
`assignLocalCosts(`$\Delta O_{old} - O_{new}, \sigma, O_{old}, O_{new}$`)`;
$O_{old} \leftarrow$ `aggregateCosts(`O_{old}`)`;
$O_{new} \leftarrow$ `aggregateCosts(`O_{new}`)`;
`transferCosts(`O_{old}, O_{new}`)`;
return O_{new};

Im ersten Schritt (*diff*) werden die Änderungen zwischen der alten (O_{old}) und neuen (O_{new}) Ontologieversion berechnet. Anschließend werden die gefundenen Änderungen sowie die Änderungskosten (σ) verwendet, um betroffenen Konzepten lokale Kosten zuzuweisen (*assignLocalCosts*). Abhängig von der Art der Änderung (*add*, *del*, *upd*) werden lokale Kosten in der alten bzw. neuen Ontologieversion erfasst. So kann beispielsweise die Löschung eines Konzepts nur in der alten Ontologieversion registriert werden, da das Konzept in der neuen Version nicht mehr vorhanden ist (umgekehrt gilt das gleiche für hinzugefügte Konzepte). Nach der Zuweisung der lokalen Kosten werden diese entlang der Ontologiestruktur nach oben (Richtung Wurzel) propagiert (*aggregateCosts*). Die dabei berechneten aggregierten Kosten fassen jeweils alle zugewiesenen lokalen Kosten aus tieferen Ontologieteilen zusammen, so dass das Wurzelkonzept der Ontologie alle zugewiesenen lokalen Kosten aggregiert. Die Propagierung wird sowohl in der alten als auch in der neuen Ontologieversion durchgeführt. Da abschließend die Erkennung der Regionen auf der neueren Ontologieversion stattfinden soll, ist ein Transfer aggregierter Kosten von O_{old} nach O_{new} (*transferCosts*) notwendig. Dieser Kostentransfer garantiert, dass entstandene Kosten in der alten Version (z. B. bei Löschungen) ebenfalls in die Bestimmung von Regionen einfließen. Nach dem Transfer wird die neue Version O_{new} mit den berechneten aggregierten Kosten zurückgegeben. Diese Version kann nun in zweierlei Hinsicht verwendet werden. Erstens ist es möglich direkt die in 6.2.2 definierten Metriken anzuwenden und damit eine Bestimmung (in)stabiler Ontologieregionen durchzuführen (siehe 6.3.2). Zweitens kann diese Version in einem iterativen Algorithmus wiederum als Eingabe zum Vergleich mit der nächst neueren Ontologieversion genutzt werden, um eine Erkennung von Regionen über mehrere Ontologieversionen zu gewährleisten (siehe allgemeiner Algorithmus in 6.3.3). In den folgenden Teilabschnitten wird der gezeigte Algorithmus *computeAggregatedCosts* für zwei Versionen detaillierter dargestellt. Dabei werden die Einzelschritte sowohl allgemein als auch anhand eines Beispiels erläutert.

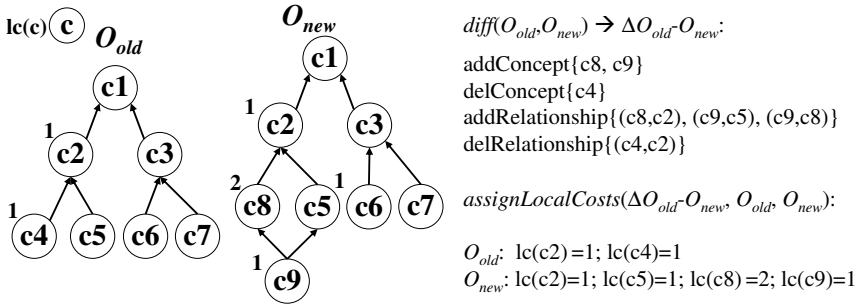


Abbildung 6.2: Diff Berechnung und Zuweisung lokaler Kosten

Änderungserkennung und Zuweisung lokaler Kosten

Die Änderungserkennung (*diff*) zwischen O_{old} und O_{new} beruht auf dem Vergleich von *accessions*, welche für die Identifikation von Konzepten verwendet werden (siehe Evolutionsmodell in Kapitel 4). Dabei werden Ontologieelemente der neuen Version mit denen der alten Version abgeglichen und entsprechende Änderungsoperationen für Konzepte, Beziehungen und Attribute erkannt. So sind eingefügte Elemente (*add*) ausschließlich in der neuen Version vorhanden, wohingegen gelöschte Elemente (*del*) nur in der alten Version auftreten. Des Weiteren werden Änderungen (*upd*) an Attributen erkannt, beispielsweise wenn der Name eines Konzepts angepasst wurde. Die mittels *diff* erkennbaren Änderungen decken somit alle gezeigten Änderungsarten in 6.2.1 ab. Die Berechnung des Diff kann ebenso auf dem komplexeren Diff-Algorithmus aus Kapitel 5 beruhen. Dies zeigt die Flexibilität des Ansatzes, d. h. in diesem Schritt können je nach Bedarf verschiedene Algorithmen zur Änderungserkennung eingesetzt werden.

Das in Abb. 6.2 dargestellte Beispiel zeigt zwei Ontologieversionen O_{old} und O_{new} mit ihren Konzepten sowie Beziehungen (aus Gründen der Übersichtlichkeit werden keine Attribute und nur is_a Beziehungen betrachtet). Die Anwendung von *diff* auf den beiden Versionen liefert das folgende Ergebnis. Es wurden zwei Konzepte $c8$ und $c9$ eingefügt (*addConcept*), wohingegen das Konzept $c4$ entfernt wurde (*delConcept*). Zugehörige Beziehungen wurden ebenfalls eingefügt ($(c8,c2),(c9,c5),(c9,c8)$) und gelöscht ($(c4,c2)$).

Die berechneten Änderungen zwischen den beiden Versionen sowie die Änderungskosten σ werden nun verwendet, um die lokalen Kosten (*lc*) der von Änderungen betroffenen Konzepte zu bestimmen. Die Methode *assignLocalCosts* geht dabei wie folgt vor. Die Zuweisung von Kosten für *add* und *upd* Änderungen erfolgt in der neuen Version O_{new}. Im Gegensatz dazu werden Kosten für Löschungen (*del*) in der alten Version O_{old} erfasst. Kosten für Änderungen an einem Konzept oder Attribut werden direkt dem entsprechenden Konzept zugewiesen. Im Falle von Änderungen an Beziehungen werden die anfallenden Kosten in den beteiligten Konzepten erfasst

(Quelle und Ziel einer Beziehung). Dabei können Quelle und Ziel jeweils unterschiedliche Kosten zugewiesen werden.

Die in Abb. 6.2 dargestellten Ziffern neben den Konzepten entsprechen den zugewiesenen lokalen Kosten auf Basis der zuvor gefundenen Änderungen. Zur Übersichtlichkeit werden im Beispiel Einheitskosten von 1 im Kostenmodell für alle Änderungen angesetzt. Für Änderungen an Beziehungen werden lediglich lokale Kosten im Zielkonzept registriert. Für das gezeigte Beispiel ergeben sich die folgenden lokalen Kosten. Da $c4$ gelöscht wurde, bekommt $c4$ in der alten Version lokale Kosten von 1 zugewiesen. Ebenso erhält $c2$ in der alten Version lokale Kosten von 1, da die Beziehung ($c4$,$c2$) entfernt wurde. Die Einfügungen von $c8$ und $c9$ führen zu lokalen Kosten 1 in beiden Konzepten. Da zusätzlich eine Beziehung zwischen $c9$ und $c8$ eingefügt wurde, werden $c8$ nochmals lokale Kosten von 1 zugewiesen, was in Summe 2 ergibt ($lc(c8) = 2$). Durch die beiden anderen Einfügungen von Beziehungen (($c9$,$c5$),($c8$,$c2$)) ergeben sich lokale Kosten von 1 für die beiden Konzepte $c2$ und $c5$.

Aggregation lokaler Kosten

Die im vorherigen Schritt zugeordneten lokalen Kosten werden nun entlang der is_a Pfade aufwärts propagiert, um die aggregierten Kosten (ac) von Konzepten zu berechnen. Die aggregierten Kosten eines einzelnen Konzepts c fassen dabei alle im is_a Subgraph von c angefallenen lokalen Kosten zusammen. Somit subsumiert die Wurzel der Ontologie alle zugeordneten lokalen Kosten in ihren aggregierten Kosten. Die Aggregation wird sowohl in der alten als auch in der neuen Ontologieversion durchgeführt.

Die Aggregation basiert auf der folgenden Regel. Die aggregierten Kosten $ac(c)$ eines Konzepts c setzen sich aus den aggregierten Kosten seiner Kinder sowie den eigenen lokalen Kosten $lc(c)$ zusammen.

$$ac(c) = \sum_{direct\ children\ c'\ of\ c} \frac{ac(c')}{|parents(c')|} + lc(c)$$

Falls ein Konzept mehrere Vorgängerkonzepte hat (Mehrfachvererbung), so werden die Kosten basierend auf der Anzahl der Vorgänger $|parents|$ gesplittet. Folglich werden jeweils Kosten $\frac{costs}{|parents|}$ an die einzelnen Vorgänger propagiert. Der nachfolgende Algorithmus $aggregateCosts$ verwendet eine Ontologieversion O_v mit assoziierten lokalen Kosten und propagiert diese mit Hilfe des $aggregate$ Algorithmus entlang der

Ontologiestruktur aufwärts:

Algorithmus 6: aggregateCosts

Input : ontology versions O_v with assigned local costs
Output : ontology version O_v with aggregated costs
forall the *concepts* $c \in O_v$ **do**
  if *local costs* $lc(c) > 0$ **then**
    `aggregate`$(c, O_v, lc(c))$;
  end
end
return O_v;

Algorithmus 7: aggregate

Input : concept c, ontology version O_v, change costs σ
aggregated costs $ac(c) \leftarrow ac(c) + \sigma$;
parent concepts $C_{parents} \leftarrow$ `getParents`(O_v, c);
normalized costs $\sigma_{norm} \leftarrow \frac{\sigma}{|C_{parents}|}$;
forall the *concepts* $c' \in C_{parents}$ **do**
  `aggregate`(c', O_v, σ_{norm});
end

Abb. 6.3 zeigt die Aggregation der lokalen Kosten für die beiden Beispielversionen O_{old} und O_{new}. Jedes Konzept wird mit seinen lokalen sowie aggregierten Kosten dargestellt $(lc(c)|ac(c))$. Die kursiven Zahlen neben den Pfaden kennzeichnen die propagierten Kosten. In der neuen Ontologieversion sieht die Aggregation wie folgt aus. Die aggregierten Kosten von $c9$ entsprechen den lokalen Kosten $lc(c9)$, da $c9$ keine Kinder besitzt. Die beiden Beziehungen ($c9$,$c5$) und ($c9$,$c8$) erlauben nun eine Propagierung der Kosten von $c9$ nach $c5$ bzw. $c8$. Da $c9$ zwei Vorgänger besitzt werden die aggregierten Kosten $ac(c9)$ in zwei Teile mit jeweils Kosten 0,5 geteilt und nach oben propagiert. Somit setzen sich die aggregierten Kosten von $c5$ wie folgt zusammen: $ac(c5) = \frac{ac(c9)}{2} + lc(c5) = 0{,}5 + 1 = 1{,}5$. Analog für $c8$: $ac(c8) = \frac{ac(c9)}{2} + lc(c8) = 0{,}5 + 2 = 2{,}5$. Danach können die aggregierten Kosten von $c5$ und $c8$ über ($c5$,$c2$) bzw. ($c8$,$c2$) nach $c2$ propagiert werden. Für die aggregierten Kosten von $c2$ ergibt dies: $ac(c2) = ac(c5) + ac(c8) + lc(c2) = 1{,}5{+}2{,}5{+}1 = 5$. Im letzten Schritt werden die aggregierten Kosten der Wurzel $c1$ berechnet. Da $c1$ keine eigenen lokalen Kosten besitzt, werden $c1$ lediglich die aggregierten Kosten von $c2$ zugewiesen: $ac(c1) = ac(c2) = 5$. Wie man sieht sind die aggregierten Kosten der Wurzel in beiden Versionen äquivalent zur Summe aller zugeordneten lokalen Kosten (2 in O_{old} und 5 in O_{new}), d. h. die Aggregation wurde korrekt ausgeführt.

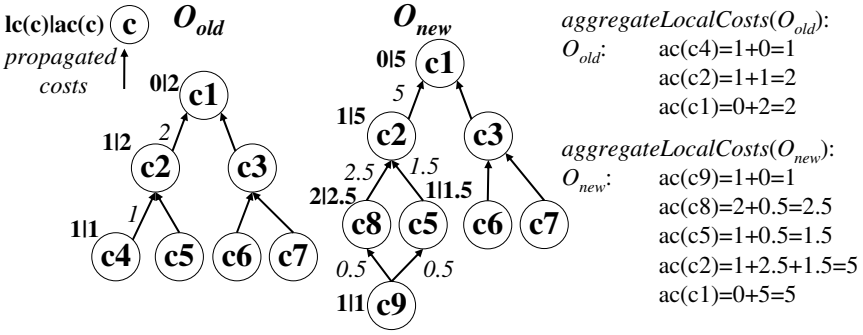


Abbildung 6.3: Aggregation lokaler Kosten

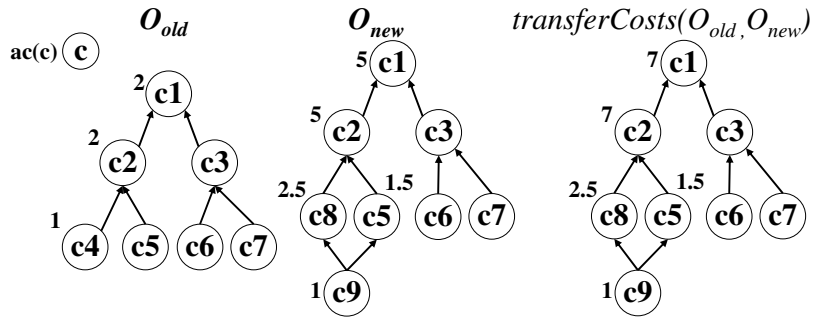
Transfer aggregierter Kosten

Die getrennte Zuweisung und Aggregation von Kosten in beiden Ontologieversionen erfordert einen Transfer der aggregierten Kosten aus der alten in die neue Ontologieversion. Dieser Transfer garantiert, dass entstandene Kosten in der alten Version ebenfalls in der neuen Version präsent sind und in die Bestimmung von Regionen mit einfließen. Der Algorithmus *transferCosts* transferiert die aggregierten Kosten aus der alten in die neue Ontologieversion. Dabei werden die aggregierten Kosten gleicher Konzepte aufsummiert:

Algorithmus 8: transferCosts

Input : ontology versions O_{old}, O_{new}
forall the *concepts* $c \in O_{old}$ **do**
  if $c \in O_{new}$ **then**
    $ac(c) \in O_{new} += ac(c) \in O_{old}$;
  end
end

Der Transfer aggregierter Kosten für das Beispiel wird in Abb. 6.4 illustriert. Die Tabelle am unteren Rand zeigt, wie die Kosten aus O_{old} nach O_{new} übertragen werden. Da die Konzepte $c1$, $c2$, $c3$, $c5$, $c6$ und $c7$ in beiden Versionen vorkommen, werden deren aggregierte Kosten aufsummiert. Beispielsweise besitzt $c2$ nach dem Transfer aggregierte Kosten von 7, welche sich aus 2 (O_{old}) und 5 (O_{new}) zusammensetzen. Für Konzepte, welche nur in der neuen Version vorhanden sind (z. B. $c8$ und $c9$), bleiben die aggregierten Kosten unverändert, d. h. es findet kein Transfer statt. Im Gegensatz dazu können aggregierte Kosten gelöschter Konzepte nicht direkt in die neue Version transferiert werden (z. B. $c4$). Allerdings wird durch die vorherige Aggregation der Kosten sichergestellt, dass Kosten aus Löschungen indirekt in die neue Version übertragen werden. Dadurch werden im Fall von $c4$ die Kosten der Löschung indirekt über das Vorgängerkonzept $c2$ in die neue Version transferiert.



$transferCosts(O_{old}, O_{new})$:

	ac(c1)	ac(c2)	ac(c3)	ac(c4)	ac(c5)	ac(c6)	ac(c7)	ac(c8)	ac(c9)
O_{old}	2	2	0	1	0	0	0		
O_{new}	5	5	0		1.5	0	0	2.5	1
transfer	7	7	0		1.5	0	0	2.5	1

Abbildung 6.4: Transfer von Änderungskosten

6.3.2 Regionenerkennung mit Metriken

Die Erkennung (in)stabiler Regionen erfolgt mit Hilfe des Algorithmus *compute-RegionMeasures*, welcher verfügbare Informationen wie aggregierte Kosten oder die Ontologiestruktur zur Auswertung der in 6.2.2 beschriebenen Metriken verwendet. Beispielsweise wird für die rel_size Metrik über alle Konzepte iteriert und der Quotient zwischen der Größe der Region und der Ontologie selbst berechnet. Dies ergibt im Beispiel eine relative Größe von 1,0 für $c1$. Die beiden Kinder $c2$ und $c3$ weisen eine relative Größe von 0,5 bzw. 0,375 auf. Das Beispiel inkl. der zugehörigen Metriken aus Abb. 6.1 entspricht dem Ergebnis des Transfers der aggregierten Kosten aus Abb. 6.4. Dadurch sind die Ergebnisse der Metriken äquivalent mit den in Abb. 6.1 gezeigten.

Für die Erkennung spezifischer Ontologieregionen werden Filterkriterien definiert und auf den Ergebnissen der Metriken angewandt. Je nach Anforderungen und Anwendung können verschiedene Filterkriterien definiert und kombiniert werden. Beispielsweise können große, stabile Regionen mit Hilfe der Filterkriterien rel_size>0,2 und avg_costs=0 selektiert werden. Im Beispiel würde die Region $c3$ die beiden Kriterien erfüllen und als große, stabile Ontologieregion klassifiziert werden. Im Gegensatz dazu könnten ebenso große, instabile Regionen mit den Filterkriterien rel_size>0,2 und avg_costs>1,0 erkannt werden. Im Beispiel würde die Region $c2$ diese Kriterien erfüllen. In der späteren Evaluierung (6.4) wird gezeigt, wie entsprechende Schwellwerte für die Filterkriterien ermittelt werden können. Um ein kompaktes Ergebnis zu gewährleisten, werden redundante Regionen, welche in anderen selektierten Regionen enthalten sind, aus dem Endergebnis entfernt. Die Region $c8$ erfüllt ebenfalls die Kriterien einer großen, instabilen Region (rel_size>0,2

und *avg_costs*>1,0), jedoch ist sie aber bereits in $c2$ enthalten. Aus diesem Grund wird lediglich die Ontologieregion $c2$ ins Endergebnis übernommen.

6.3.3 Allgemeiner Algorithmus für mehrere Versionen

Auf Basis der vorgestellten *computeAggregatedCosts* und *computeRegionMeasures* Algorithmen wird nun der generelle Algorithmus *findRegions* für mehrere Ontologieversionen vorgestellt. Der komplette *findRegions* Algorithmus sieht wie folgt aus:

Algorithmus 9: findRegions

Input : ontology versions $O_1, \ldots, O_n$, change costs σ
forall the *succeeding versions* $O_i - O_{i-1}$ **do**
  $O_{i+1} \leftarrow$ `computeAggregatedCosts(`O_i, O_{i+1}, σ`);`
end
`computeRegionMeasures(`O_n`);`

Der Algorithmus basiert auf der folgenden Idee: Für n veröffentlichte Ontologieversionen $(O_1, \ldots, O_n)$ wird über die einzelnen Versionen iteriert und der *computeAggregatedCosts* Algorithmus auf jedem Paar (O_i, O_{i+1}) ausgeführt. Somit werden alle Änderungen zwischen aufeinanderfolgenden Versionen erfasst und die berechneten aggregierten Kosten werden sukzessive in die letzte Version O_n transferiert. In der letzten Version kann nun die Bestimmung von Regionen mit Hilfe des *computeRegionMeasures* Algorithmus durchgeführt werden. Der allgemeine Algorithmus wird in der folgenden Evaluierung verwendet, um (in)stabile Regionen in großen Ontologien der Lebenswissenschaften zu identifizieren.

6.4 Evaluierung

Der Ansatz wurde anhand zweier weit verbreiteter und genutzter Ontologien evaluiert: Gene Ontology (GO) sowie der National Cancer Institute Thesaurus (NCIT). In den nachfolgenden Unterabschnitten wird zunächst das Setup der Evaluierung näher erläutert. Danach erfolgt eine erste vergleichende Analyse der Gesamtstabilität beider Ontologien. Im Anschluss wird eine Verteilungsanalyse von Ontologieregionen bzgl. ihrer Stabilität präsentiert. Dabei wird erläutert wie die stabilsten bzw. instabilsten Regionen selektiert werden können. Abschließend wird gezeigt, wie der Algorithmus zum Tracking der Stabilitäten von Regionen über längere Zeiträume hinweg verwendet werden kann.

6.4.1 Setup

Die zwei zu untersuchenden Ontologien werden in diversen Projekten bzw. Applikationen genutzt und unterliegen ständigen Aktualisierungen. GO wird beispielsweise zur einheitlichen Funktionsbeschreibung (Annotation) von Proteinen bzgl. Biologischen Prozessen (BP), Molekularen Funktionen (MF) und Zellulären Komponenten (CC) eingesetzt. NCIT besteht aus 20 Hauptkategorien, welche krebsbezogene Themen wie Medikamente, Anatomie oder Gewebe klassifizieren und beschreiben. Der Thesaurus wird in US-weiten Projekten wie dem Cancer Biomedical Informatics Grid (caBIG) [18] verwendet.

Für die Evaluierung wurden veröffentliche Ontologieversionen zwischen 2004 und 2009 in ein Repository integriert und versioniert (Details zum Repository und zur Versionierung siehe Kapitel 8). Pro Monat wurde maximal eine Ontologieversion integriert, falls mehrere Versionen vorlagen wurde die erste Version des Monats verwendet. Das Repository ermöglicht einen direkten Zugriff auf integrierte Versionen sowie deren Vergleich für die Änderungserkennung. Die letzte GO Version von Dezember 2009 beinhaltet 30.304 Konzepte (GO-BP: 18.108, GO-MF: 9.459, GO-CC: 2.737). NCIT hingegen weist in seiner Dezember Version eine Konzeptanzahl von 77.465 auf.

Für die Evaluierung wurde das folgende Kostenmodell verwendet:

concept		**relationship**		**attribute**		
add	*del*	*add*	*del*	*add*	*del*	*upd*
1,0	2,0	1,0	2,0	0,5	0,5	0,5

Generell haben Änderungen an Konzepten den größten Einfluss, gefolgt von Änderungen an Beziehungen und Attributen. Löschungen von Konzepten bzw. Beziehungen wird ein höherer Einfluss zugewiesen, wohingegen Änderungen an Attributen gleich bewertet werden. Bei Änderungen an Beziehungen wird eine Hälfte der Kosten an das Quellkonzept und die andere Hälfte an das Zielkonzept übertragen. Die verwendeten Kosten im Modell basieren auf den Erfahrungen aus vorherigen Arbeiten (siehe Kapitel 4), je nach Anforderungen und Anwendung kann das Kostenmodell entsprechend variiert und angepasst werden.

6.4.2 Gesamtstabilität der Ontologien

Um die Gesamtstabilität der Ontologien beurteilen zu können, werden die Metriken auf der jeweiligen Wurzel *root* angewandt, d. h. die gesamte Ontologie wird als eine Region aufgefasst und mit Hilfe der Metriken bzgl. Stabilität bewertet. Im Nachfolgenden werden die Metriken $abs\_size(root)$, $abs\_costs(root)$ sowie $avg\_costs(root)$

	abs_size(root)		*abs_costs(root)*		*avg_costs(root)*	
	2008	*2009*	*2008*	*2009*	*2008*	*2009*
GO	27,799	30,304	24,242	19,412	0.87	0.64
– MF	9,205	9,459	4,636	3,002	0.50	0.32
– BP	16,231	18,108	17,594	14,557	1.08	0.80
– CC	2,363	2,737	2,011	1,854	0.85	0.68
NCIT	71,337	77,455	23,165	36,562	0.32	0.47

Tabelle 6.1: Gesamtstabilität der untersuchten Ontologien in 2008 und 2009

zur Einschätzung der Gesamtstabilität herangezogen. Tab. 6.1 zeigt die Gesamtstabilität für GO und ihrer Subontologien sowie NCIT für 2008 und 2009.

In 2008 weisen GO mit ≈24.200 und NCIT mit ≈23.200 ähnliche absolute Kosten auf. Hingegen variieren die durchschnittlichen Änderungskosten pro Konzept; GO zeigt deutlich erhöhte Kosten von 0,87 gegenüber 0,32 für NCIT. In der zweiten Periode (2009) stiegen die absoluten Kosten für NCIT an, während die Kosten für GO sanken. GO bleibt jedoch im Durchschnitt (avg_costs) änderungsintensiver als NCIT (0,64 für GO und 0,47 für NCIT). Innerhalb der GO Subontologien besitzen die Biologischen Prozesse (BP) die größten absoluten sowie durchschnittlichen Änderungskosten in beiden Perioden. Umgekehrt können die Molekulare Funktionen (MF) als die stabilste Subontologie von GO angesehen werden ($\leq 0,5$ avg_costs in 2008 und 2009). Insbesondere sanken die durchschnittlichen Änderungskosten von GO-MF von 0,5 auf 0,32 zwischen 2008 und 2009 was für eine erhöhte Stabilität gegenüber GO-CC und GO-BP spricht.

6.4.3 Bestimmung (in)stabiler Ontologieregionen

Um die (in)stabilsten Regionen einer Ontologie zu ermitteln, wird zunächst eine Verteilungsanalyse aller vorhandenen Regionen bzgl. ihrer durchschnittlichen Änderungskosten (avg_costs) durchgeführt. Abb. 6.5 zeigt beispielhaft eine solche Verteilung für GO-BP in 2009. In der Analyse werden Ontologieregionen mit einer Mindestgröße von 0,3% erfasst (ca. 50 Konzepte im Falle von GO-BP). Die Regionen werden auf Basis ihrer durchschnittlichen Kosten in Intervalle der Größe 0,05 gruppiert. Insgesamt wurden für GO-BP 518 Regionen in 36 Intervalle (0,00:0,05 bis 1,75:1,80) klassifiziert. Die meisten Regionen (≈430; 80%) weisen durchschnittliche Kosten zwischen 0 und 0,5 auf. Davon besitzen ca. 60 (≈12%) Kosten unter 0,05 und können daher als sehr stabil angesehen werden. Im Gegensatz dazu weisen ≈10% der Regionen überdurchschnittliche Kosten von mehr als 0,65 auf und sind daher als stark instabil einzuschätzen.

Auf Basis der vorherigen Analyse können die instabilsten bzw. stabilsten Regionen an den Rändern der Verteilung bestimmt werden. Je nach Bedarf und Anforderun-

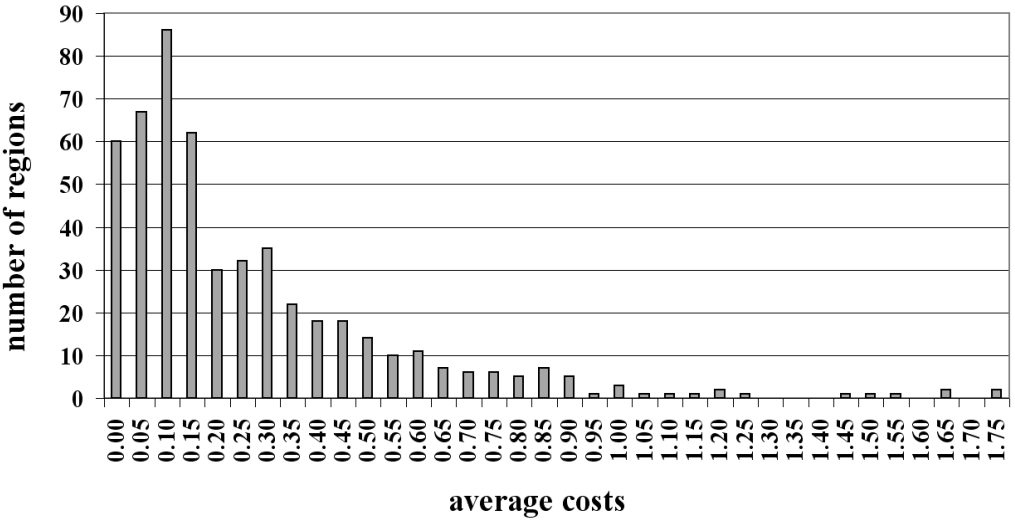


Abbildung 6.5: Verteilung der Regionen in GO-BP nach ihren durchschnittlichen Kosten

gen können absolute Schwellwerte (z. B. $avg\_costs < 0{,}01$ oder $avg\_costs > 0{,}8$) oder Perzentile der Verteilung für die Klassifikation (in)stabiler Regionen verwendet werden. In der folgenden Analyse werden alle Regionen unterhalb des 5%-Perzentils als stabil klassifiziert; Regionen oberhalb des 95%-Perzentils werden als instabil erachtet.

Tab. 6.2 zeigt die sechs größten (in)stabilen Regionen für GO und NCIT in 2009. Die relative Größe der Regionen variiert zwischen 0,5% und 5% der Gesamtontologiegröße. Im Fall von GO sind die sechs instabilsten Regionen größer als die sechs stabilsten Regionen. So hat die größte stabile Ontologieregion eine Größe von 1,32% (GO:0031300), wohingegen die sechst größte instabile Region (GO:0048646) eine Größe von 1,4% besitzt. NCIT besitzt mit mehr als 400 Konzepten pro Region die absolut größten stabilen Regionen. Keine der als stabil klassifizierten Regionen von NCIT hat irgendwelche Änderungen erfahren (avg_costs=0). In GO hingegen zeigen einige der als stabil bestimmten Regionen geringe durchschnittliche Kosten (z. B. GO:0050865 und GO:0075136). Einer starken Änderungsintensität unterlagen insbesondere Themengebiete wie anatomische Strukturen („anatomical structure“) z. B. GO:0009653, GO:0048856 oder GO:0048646. Auch spezielle Bindungsfunktionen wie „receptor binding“ oder „nucleic acid binding“ wurden stark bearbeitet. So ist beispielsweise GO:0005102 („receptor binding“) die größte instabile Region in GO (rel_size=4,31%). In NCIT ist „Retired Concept“ mit einer relativen Größe von 4,21% die größte instabile Region. In dieser Region werden alle veralteten und nicht mehr benötigten Konzepte von NCIT gesammelt, wodurch diese hohe Änderungsintensität zu erklären ist. Andere änderungsintensive Regionen in NCIT betreffen Themen aus dem Bereich „Drugs and Chemicals“ wie z. B. „Industrial Aid“ oder „Natural Product“.

		accession	name	abs_size	rel_size	avg_costs
GO	unstable	GO:0005102	receptor binding	408	4.31%	0.95
		GO:0009653	anatomical structure morphogenesis	583	3.22%	1.22
		GO:0048856	anatomical structure development	566	3.13%	0.91
		GO:0033643	host cell part	77	2.81%	1.90
		GO:0003676	nucleic acid binding	241	2.55%	0.86
		GO:0048646	anatomical structure formation involved in morphogenesis	253	1.40%	0.92
	stable	GO:0031300	intrinsic to organelle membrane	36	1.32%	0.000
		GO:0030054	cell junction	31	1.13%	0.000
		GO:0050865	regulation of cell activation	184	1.02%	0.012
		GO:0075136	response to host	181	1.00%	0.019
		GO:0000151	ubiquitin ligase complex	25	0.91%	0.000
		GO:0016860	intramolecular oxidoreductase activity	71	0.75%	0.000
NCIT	unstable	C28428	Retired Concept	3,264	4.21%	3.49
		C53791	Adverse Event Associated with Infection	1,186	1.53%	2.36
		C45678	Industrial Aid	889	1.15%	1.40
		C74944	Clinical Pathology Procedure	747	0.96%	0.84
		C66892	Natural Product	708	0.91%	1.35
		C53543	Rare Non-Neoplastic Disorder	504	0.65%	1.22
	stable	C64389	Genomic Feature Physical Location	1,026	1.32%	0.000
		C23988	Mouse Neoplasms	886	1.14%	0.000
		C48232	Cancer TNM Finding	742	0.96%	0.000
		C53798	Adverse Event Associated with Surgery & Intra-Operative Injury	707	0.91%	0.000
		C43877	American Indian	555	0.72%	0.000
		C53832	Infection Adverse Event with Unknown Absolute Neutrophil Count	386	0.50%	0.000

Tabelle 6.2: Größte (in)stabilste Regionen in 2009

6.4.4 Tracking der Stabilität von Ontologieregionen

Eine beispielhafte Anwendung des vorgestellten Algorithmus ist das Tracking der Änderungsintensitäten von Regionen über längere Zeiträume. Der Algorithmus wird dabei auf verschiedene Zeitintervalle (-perioden) einer Ontologie angewandt. Die berechneten Änderungsintensitäten können dann über die Zeit verfolgt und eingeschätzt werden. So können u. a. Trends in der Evolution von Ontologien erkannt werden. Beispielsweise kann sich ein Nutzer die Frage beantworten lassen, welche Regionen wann ein starkes Interesse hervorgerufen haben oder nicht.

Als Anwendungsfall wurde ein Tracking in NCIT zwischen 2004 und 2009 durchgeführt. Als zu untersuchende Regionen wurden die 20 Hauptkategorien des NCIT verwendet, da diese den Thesaurus grob in Themengebiete wie „Drugs and Chemicals“ oder „Anatomic Structure“ einteilen. Die Berechnung erfolgte mit Hilfe eines Zeitfensters („Sliding Window“), welches wie folgt angewandt wurde. Das Zeitfenster umfasst ein halbes Jahr und wurde beginnend in 2004 mit einer Schrittweite von einem Monat sukzessive bis Ende 2009 über den gesamten Analysezeitraum geschoben. Für jeden Schritt dienten die im Zeitfenster veröffentlichten Versionen als Eingabe für den Algorithmus. Die jeweils berechneten Änderungsintensitäten der entsprechenden Regionen wurden erfasst und konnten somit in einer Trendanalyse verwendet werden. In den erfassten Intensitäten können somit u. a. Varianzen / Abweichungen analysiert und interpretiert werden, z. B. wo und wann starke Änderungen aufgetreten sind oder nicht.

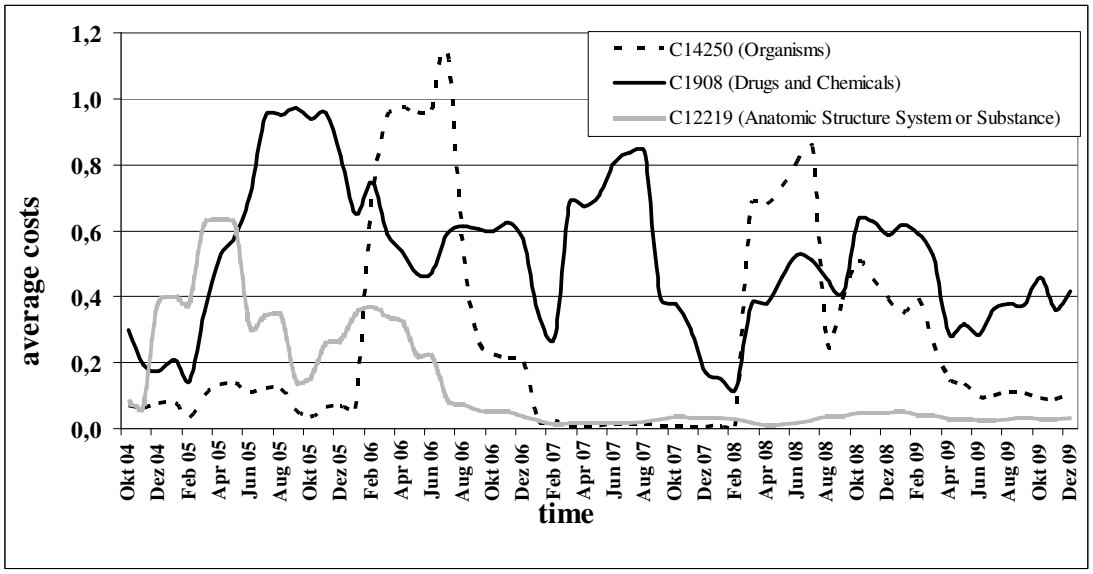


Abbildung 6.6: Tracking der Änderungsintensität für ausgewählte Regionen aus NCIT

Das in Abb. 6.6 dargestellte Diagramm zeigt das Tracking der durchschnittlichen Kosten für drei ausgewählte Hauptkategorien des NCIT zwischen 2004 und 2009. Es können verschiedene Muster der Evolution ausgemacht werden. Erstens existieren Regionen wie „Drugs and Chemicals“, welche ständig erhöhte Änderungskosten aufweisen. Diese Regionen erfahren fortlaufend Änderungen an ihrem Inhalt, da sie aktive Forschungsfelder repräsentieren und werden in naher Zukunft wahrscheinlich weiterhin diese erhöhte Änderungsintensität zeigen. Zweitens treten Regionen mit Perioden erhöhter Instabilität (Änderungsintensität) gepaart mit längeren stabilen Perioden auf. Die Region „Organisms“ unterlag zwischen März 2006 und Februar 2007 sowie März 2008 und März 2009 zahlreichen Änderungen (hohe durchschnittliche Änderungskosten). Diese Spitzen der Änderungsintensität können eine Folge neuer Forschungserkenntnisse sein, welche nach und nach in die Ontologie integriert worden sind. Anderseits kann die Projektkoordination die Ontologieentwicklung steuern, d. h. Bereiche festlegen an denen intensiv gearbeitet werden soll oder Restrukturierungen durchzuführen sind. Drittens wurden Regionen erkannt, welche mit fortlaufender Zeit an Stabilität zunahmen. So wurde die Region „Anatomic Structure System or Substance“ zu Beginn des Analysezeitraums (bis ca. Ende 2006) stark weiterentwickelt und angepasst. Seither ist eine Stabilisierung der Änderungsintensität erkennbar (avg_costs<0,05), d. h. es werden lediglich kleinere Verfeinerungen/Korrekturen durchgeführt und die Region kann als nahezu finalisiert angesehen werden. Die Wahrscheinlichkeit für künftige dramatische Änderungen innerhalb solcher Regionen ist eher gering. Diese Beobachtung gilt insbesondere für Regionen, welche ein bereits akzeptiertes Standardwissen (z. B. Anatomie des Menschen in den Lebenswissenschaften) modellieren.

6.5 Zusammenfassung

Dieses Kapitel führte die Notation von Ontologieregionen und zugehörigen Metriken zur Beurteilung ihrer Änderungsintensität (Stabilität) ein. Der vorgestellte automatisierte Algorithmus erlaubt die Bestimmung (in)stabiler Ontologieregionen auf Basis veröffentlichter Ontologieversionen in einem vorgegebenen Zeitraum. Dabei werden die Änderungen (Diff) zwischen den Versionen betrachtet, um mit Hilfe der Ontologiestruktur änderungsintensive bzw. stabile Ontologieregionen zu bestimmen. Der Algorithmus verwendet ein adaptierbares Kostenmodell, welches den Einfluss von Änderungen auf die Ontologie beurteilt und zur Bestimmung der Regionen genutzt wird. Der vorgestellte Ansatz kann in verschiedenen Szenarien Anwendung finden. Nutzer von Ontologien können a-priori abschätzen, ob eine Neuberechnung ihrer Ontologie-basierten Analysen oder Experimente auf Basis einer neueren Ontologieversion als sinnvoll erscheint oder nicht. Weiterhin können Ontologiedesigner die Entwicklung ihrer Ontologie beobachten und beurteilen, um beispielsweise künftige Arbeiten und Änderungen in Ontologieteilen zu planen und zu koordinieren. Der Ansatz wurde anhand zweier intensiv genutzter und häufig modifizierter Ontologien aus den Lebenswissenschaften evaluiert. Eine vergleichende Evaluierung zeigte, dass der Algorithmus in der Lage ist (in)stabile Ontologieregionen automatisiert zu bestimmen. Das Tracking der Änderungsintensität von Regionen über längere Zeiträume erlaubt die Abschätzung von Entwicklungstrends innerhalb der verschiedenen Ontologieteile. Im Rahmen der Evaluierung wurden unterschiedliche Entwicklungsmuster identifiziert. Dazu zählen beispielsweise Regionen, welche ständig Änderungen unterworfen sind oder jene die in den letzten Jahren mehr und mehr an Stabilität gewonnen haben.

Teil III

Systeme zur Analyse der Ontologieevolution

7

Webapplikation – Ontology Evolution Explorer

7.1 Motivation

Die Analysen in Kapitel 4 zeigten, dass Ontologien in den Lebenswissenschaften kontinuierlich verändert und angepasst werden. Das Resultat ist eine Serie von Ontologieversionen, welche Endnutzern zur Verfügung gestellt werden. So verdoppelten sich beispielsweise die Größe der Gene Ontology bzw. des NCI Thesaurus in den letzten fünf Jahren (GO: 13.163 → 28.250, NCI: 28.740 → 68.862). Neben der dominierenden Anzahl von Einfügungen, welche zum Wachstum der Ontologien führten, existieren auch andere Änderungen wie das Löschen oder „Obsolete"-Setzen von Konzepten (ca. 25 bzw. 5 Änderungen dieser Art pro Monat in GO bzw. NCI Thesaurus). Dabei kann das „Obsolete"-Setzen eines Konzepts als Alternative zum direkten Löschen angesehen werden. Somit resultieren beide Operationen in einer ähnlichen Informationsänderung an der Ontologie und sind Indikator für eine geringe Ontologiestabilität. Daraus resultieren negative Effekte in der Ontologienutzung, z. B. sind Annotationen, welche ein gelöschtes oder veraltetes Konzept verwenden, nicht mehr gültig und müssen ebenfalls gelöscht oder angepasst werden. Auch können Einfügungen neuer Konzepte die Erzeugung neuer Annotationen zur Folge haben. Somit besteht ein Bedarf Ontologienutzer über die Evolution der verwendeten Ontologien zu informieren. Da Änderungen nicht alle Teile einer Ontologie betreffen und auch eine unterschiedliche Frequenz besitzen, ist es zusätzlich sinnvoll eine Funktionalität zur Beurteilung und Begutachtung der Evolution einzelner Konzepte anzubieten,

beispielsweise wie haben sich der Name eines Konzepts oder dessen Elternkonzepte in der Vergangenheit geändert. Diese Informationen sind speziell für Kuratoren wertvoll, welche an der manuellen Generierung von Annotationen arbeiten. Auch Forscher, welche in ihren Analysen (Auswertungen) bestimmte Teile einer Ontologie nutzen, können Informationen über die Evolution entsprechend einfließen lassen. Des Weiteren würde eine semi-automatische Anpassung (Migration) veralteter Annotationen an eine neue Ontologieversion den enormen Zeit- bzw. Ressourcenaufwand einer manuellen Anpassung stark reduzieren.

Bisherige Arbeiten auf dem Gebiet der Ontologieevolution in den Lebenswissenschaften fokussierten vorzugsweise auf die Durchführung von Änderungen (Change Management) oder der Ontologieversionierung (siehe Kapitel 2 zu Verwandten Arbeiten). Werkzeuge wie Protégé [65], KAON [119] oder OBO Edit [25] basieren zudem auf speziellen Ontologieformaten wie OBO, OWL oder RDF. Sie berücksichtigen dabei jedoch nicht den Einfluss von Ontologieänderungen auf Ontologienutzende Daten wie Annotationen oder die Ergebnisse von Ontologie-basierten Analysen. Informationen über Änderungen an Ontologien (d. h. die Rückverfolgung von Änderungen) sind auf Mailinglisten oder vereinzelte Berichte der Ontologieprovider limitiert. Beispielsweise erzeugt das GO Konsortium einen monatlichen Bericht[23], welcher die angefallenen Änderungen in GO zusammenfasst. Mailinglisten informieren interessierte Nutzer über Änderungen und Modifikationen an OBO Ontologien[24]. Die genannten Informationsquellen beschreiben die angefallenen Änderungen textuell und können dadurch nicht direkt für eine automatische Weiterverarbeitung, wie z. B. die Migration von Annotationen, verwendet werden.

Es existieren zahlreiche Online-Tools, welche einen schnellen und benutzerfreundlichen Zugang zu Ontologieinformationen bzw. verwandten Daten anbieten. Die bekannten Browser AmiGO [19] sowie QuickGO[25] ermöglichen das Anfragen sowie die Suche innerhalb der Gene Ontology. Dies umfasst die Konzepte der GO selbst wie auch assoziierte Genprodukte. Ergebnisse werden mit Hilfe der Graphstruktur von GO präsentiert. Des Weiteren können Nutzer gefundene Ergebnisse für eine Weiterverarbeitung filtern und abspeichern. Andere Onlineportale verfolgen eine Integration verschiedener Ontologien und bieten somit einen einheitlichen Zugang zu allen integrierten Ontologien an. So bietet der Ontology Lookup Service (OLS) des EBI [20] einen webbasierten Zugang zu ca. 60 Ontologien der Lebenswissenschaften über ein zentrales Portal an. Der Terminology Browser des NCI[26] verfügt über eine zentrale Schnittstelle zu Ontologien wie dem NCI Thesaurus, GO oder SNOMED CT [29]. Die Funktionalität all dieser Systeme ist jedoch auf das Durchsuchen und Browsen einer Version einer Ontologie limitiert, d. h. Nutzer können in der Regel nur auf die letzte (aktuelle) Version interaktiv zugreifen. Informationen aus der Historie

[23] http://www.geneontology.org/MonthlyReports/
[24] https://lists.sourceforge.net/lists/listinfo/obo-diffs
[25] http://www.ebi.ac.uk/QuickGO/
[26] http://nciterms.nci.nih.gov/

der Ontologie, z. B. über frühere Versionen oder Änderungen an Konzepten werden nur selten zur Verfügung gestellt. Beispielsweise gibt der NCI Terminology Browser lediglich Informationen über Erzeugungs- bzw. Änderungsereignisse von Konzepten an, aber nicht wie genau sich die Konzepte geändert haben.

Im Gegensatz zu webbasierten Ontologiebrowsern existieren nur wenige Online-Tools oder Web-Services, welche einen Vergleich von Ontologieversionen zur Änderungserkennung anbieten. Das OWS Framework [24] schlägt eine Web-Service-basierte Architektur für den Zugriff und die Modifikation von Ontologien vor. Der Diff Web-Service berechnet dabei für zwei OWL-basierte Ontologieversionen einen strukturellen Diff (Konzepte, welche eingefügt, gelöscht oder verändert wurden). Ein weiterer Web-Service zum Vergleich von Ontologieversionen wird in [121] beschrieben. Der Service berechnet für zwei Ontologieversionen in OWL deren semantische Differenz, welche über einen numerischen Wert ausgedrückt wird. GOChase [99] ist eines der wenigen Online-Tools, welches neben den üblichen Such- und Browsing-Funktionalitäten ebenfalls die Möglichkeit eines Zugangs zu historischen Ontologieinformationen bietet. Das Tool ist spezifisch für GO, wird allerdings seit einiger Zeit nicht mehr aktualisiert und ist auch nicht mehr abrufbar[27]. Die in GOChase geführten Historien basieren auf den Accessions der GO Konzepte, welche es dem Tool ermöglicht inkorrekte Links zu GO Konzepten zu identifizieren.

Wie zuvor dargestellt, existiert derzeit kein Online-Tool, welches eine ad-hoc Evolutionsanalyse für Ontologien in den Lebenswissenschaften bietet. Aus den genannten Gründen wird in diesem Kapitel ein neuartiges System OnEX (Ontology Evolution Explorer – `http://www.izbi.de/onex`) zur webbasierten Evolutionsanalyse von Ontologien und zugehörigen Versionen präsentiert. Das System adressiert vorrangig Bioinformatiker sowie Biologen, die mit Ontologien bzw. Ontologie-nutzenden Daten arbeiten. Die Beiträge des Kapitels sind die Folgenden:

- Präsentation der 3-Schichtenarchitektur von OnEX bestehend aus Repository, Middleware und webbasiertem Frontend
- Vorstellung der drei Workflows in OnEX, welche verschiedene Aspekte der Evolutionsanalyse adressieren: (1) Quantitative Evolutionsanalyse zur Beurteilung der Gesamtevolution einer Ontologie, (2) Konzept-basierte Analyse zur Untersuchung von Änderungen an Konzepten und (3) Annotation-Migration zur semi-automatischen Migration veralteter Annotationen
- Darstellung des aktuellen Inhalts von OnEX und Diskussion der erzielten Resultate

[27] `http://www.snubi.org/software/GOChase/`

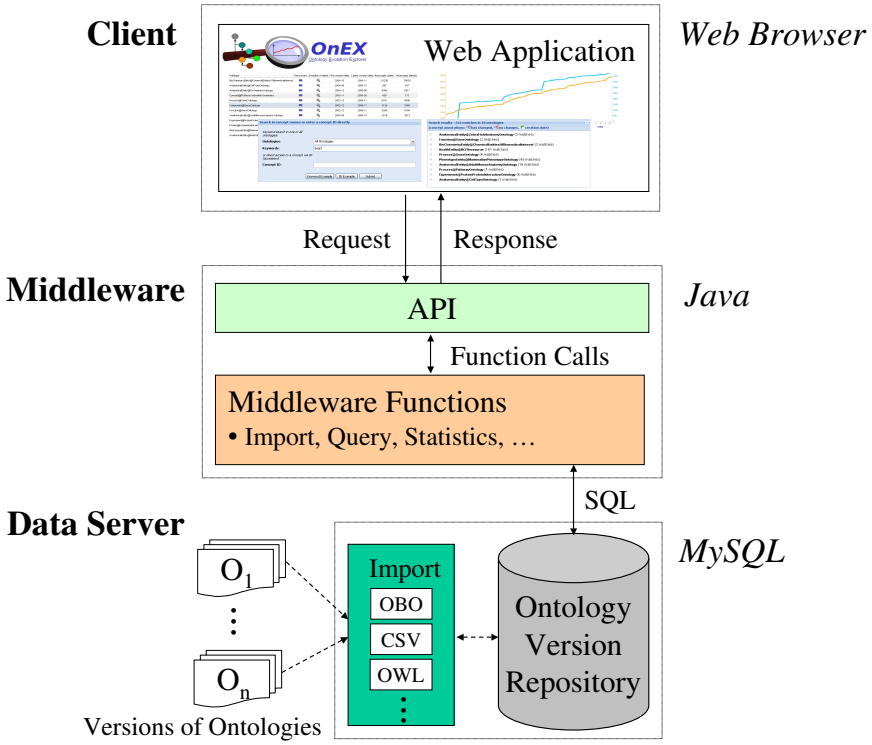


Abbildung 7.1: Architektur des OnEX Systems

7.2 Architektur und Aufbau

Das OnEX System basiert auf einer 3-Schichtenarchitektur, welche in Abb. 7.1 dargestellt ist. Das Backend des Systems besteht aus einer relationalen Datenbank (MySQL Server). Das Repository ermöglicht das Management mehrerer Ontologieversionen sowie die Sicherung zugehöriger Evolutionsstatistiken. Eine flexible Integration von Ontologien verschiedener Formate und ihrer Versionen wird durch Import-Module sichergestellt. Die in Java realisierte Middleware-Komponente besteht aus einer API, welche von Applikationen oder anderen Komponenten/Services genutzt werden kann. Somit werden zentrale Funktionalitäten wie Anfragemethoden, welche über SQL mit dem Repository interagieren, bereit gestellt. Das Frontend, d. h. die Webapplikation selbst, ist plattformunabhängig und kann somit in verschiedenen Browsern verwendet werden. Es basiert auf dem Google Web Toolkit[28] sowie der Ext GWT Library[29], welche eine Gestaltung interaktiver Webapplikationen ermöglichen. Innerhalb des Frontends sind die drei Workflows zur online Evolutionsanalyse von Ontologien verfügbar (siehe Abschnitt 7.4). Die Ergänzung einer Web-Service-Schnittstelle für einen programmatischen Zugang zu den Analyseroutinen bzw. den

[28]Google Web Toolkit: `http://code.google.com/webtoolkit/`
[29]Ext GWT Library: `http://extjs.com/products/gxt`

Ontologieversionen ist möglich.

7.3 Import und Versionierung von Ontologien

Die Import-Funktionalität nutzt die öffentlich zugänglichen Archive der Ontologieprovider, z. B. die CVS Verzeichnisse der OBO Foundry[30] oder das Archiv der Gene Ontology[31]. Die in den Archiven verfügbaren Ontologieversionen können über diverse Import-Module in das Repository integriert werden. Für Details zur Versionierung der Ontologien innerhalb des Repositories sowie der Integration der Versionen wird auf das nachfolgende Kapitel 8 verwiesen. Es wird das verwendete Versionierungsmodell vorgestellt und speziell darauf eingegangen wie effizient die zahlreichen Versionen im Repository verwaltet werden.

Das Repository wird durch automatische Routinen periodisch aktualisiert. Die Routinen prüfen, ob eine aktuellere Ontologieversion vorliegt und integrieren diese aus dem Archiv der Ontologieprovider. Da nur wenige Ontologieprovider täglich eine neue Ontologieversion veröffentlichen (z. B. GO) wird maximal eine Version pro Ontologie und Monat integriert. Falls mehrere Versionen in einem Monat veröffentlicht wurden, wird lediglich die erste verfügbare des Monats integriert. Natürlich werden trotzdem alle Änderungen zwischen den veröffentlichten Versionen erfasst, da die auf Monatsbasis integrierten Versionen verglichen werden. Das Repository zur Verwaltung der Versionen ist nicht auf ein bestimmtes Ontologieformat wie OBO limitiert, d. h. es können mittels der flexiblen Importer Ontologien unterschiedlicher Formate integriert werden (z. B. OWL oder CSV).

7.4 Szenarien und Workflows

OnEX ermöglicht die Analyse und Untersuchung der Ontologieevolution durch drei interaktive Workflows:

(1) *Quantitative Evolutionsanalyse* für ganze Ontologien

(2) *Konzept-basierte Analyse* zur Exploration von Änderungen an Konzepten

(3) *Annotation-Migration* zur Anpassung veralteter Annotationen an neuere Ontologieversionen

Die drei Workflows werden in den folgenden Unterabschnitten näher erläutert und diskutiert.

[30] `http://obo.cvs.sourceforge.net/`
[31] `http://archive.geneontology.org/`

7.4.1 Quantitative Evolutionsanalyse

Die Quantitative Evolutionsanalyse bietet dem Nutzer einen kompakten Überblick über die Evolution der in OnEX erfassten Ontologien. Eine Überblicktabelle fasst vergleichend Änderungsstatistiken sowie Informationen über die Entwicklung der Größe einer Ontologie zusammen. Für eine Ontologie können Trendcharts erzeugt werden, welche die Historie der Evolution graphisch darstellen. Weitere Tabellen ermöglichen die Anzeige von Änderungen zwischen den Versionen einer Ontologie: Einfügungen, Löschungen, Obsolete-Änderungen und Zusammenfassungen von Konzepten. Mit Hilfe der verfügbaren Komponenten können Nutzer zuerst eine vergleichende Analyse der Evolution zwischen den Ontologien durchführen. Anschließend besteht die Möglichkeit sich auf eine konkrete Ontologie zu fokussieren und deren Evolution zu untersuchen. Somit ist es möglich Phasen der Instabilität bzw. Stabilität einer Ontologie zu ermitteln. Dies ist insbesondere wichtig, wenn der Einfluss von Ontologieänderungen auf Ontologie-basierte Daten wie Annotationen oder Analyseergebnisse abgeschätzt werden muss. Zudem besteht die Möglichkeit schnell neue bzw. veränderte Konzepte innerhalb einer Ontologie zu lokalisieren.

Eine konkrete Anwendung des Workflows ist in Abb. 7.2 dargestellt. Der Workflow zeigt die Untersuchung der Evolution der Subontologie Biologische Prozesse in Gene Ontology (GO-BP). Das Übersichtsfenster (*Comparative Overview*) stellt alle Basisstatistiken überblicksartig in einer Tabelle dar. So ist beispielsweise ersichtlich, dass GO-BP über ≈19.000 Konzepte verfügt, welche durch ≈39.000 Beziehungen miteinander verknüpft sind (Stand März 2010). Im Gegensatz dazu wies die Ontologie in der ersten verfügbaren Version von Dezember 2002 lediglich ≈7.000 Konzepte auf. Beziehungen gab es in dieser Version nicht, d. h. die Ontologie war zu diesem Zeitpunkt noch unstrukturiert und besaß somit auch keine Graphstruktur. Der *Trend Chart* für GO-BP zeigt einen stetigen Anstieg in der Anzahl der Konzepte bzw. Beziehungen. Es ist ersichtlich, dass Beziehungen erstmalig in der Version von April 2003 eingeführt wurden. Zudem ist ein enormer Zuwachs zwischen Juli und Dezember 2006 auszumachen.

Nach einem ersten Eindruck über die Evolution von GO-BP innerhalb des gesamten Analysezeitraums können Nutzer nun zu weiteren Details navigieren. Die *Evolution Details* stellen Informationen wie die durchschnittliche Anzahl von Änderungen sowie die Anzahl von Änderungen zwischen den Ontologieversionen zur Verfügung. So besitzt GO-BP im Durchschnitt 130 Einfügungen von Konzepten pro Monat. Im Gegensatz dazu existieren durchschnittlich 12 Änderungen an Konzepten pro Monat. Die exakte Anzahl an Einfügungen, Löschungen, Obsolete-Markierungen bzw. Zusammenfassungen von Konzepten zwischen den Ontologieversionen werden in einer separaten Tabelle dargestellt. Die Tabelle ermöglicht eine Sortierung nach verschiedenen Kriterien wie z. B. die Anzahl betroffener Konzepte. So können Nutzer schnell kritische Versionsübergänge ausmachen oder zu Versionen navigieren welche sie interessieren. Für GO-BP kann man u. a. erkennen, dass die meisten Einfügungen

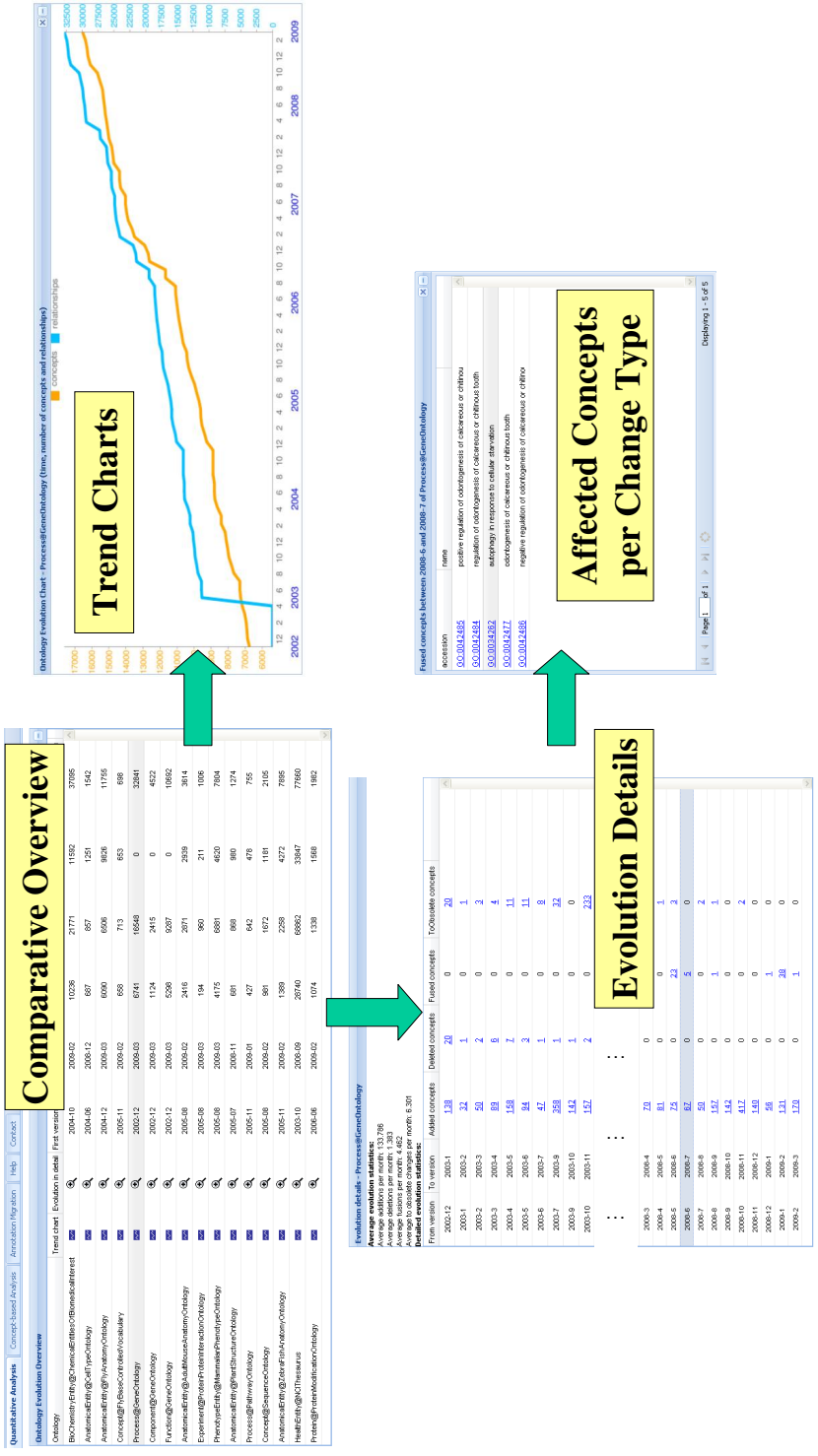


Abbildung 7.2: Workflow Quantitative Evolutionsanalyse am Beispiel GO-BP

zwischen November und Dezember 2006 stattfanden (971 Konzepte). Interessiert sich der Nutzer für einen konkreten Versionsübergang, kann zu einer weiteren Sicht *Affected Concepts per Change Type* navigiert werden. Das Fenster listet alle betroffenen Konzepte einer bestimmten Änderung zwischen zwei Versionen auf. So wurden in der Juli Version von GO-BP fünf Konzepte als veraltet (obsolete) markiert, z. B. GO:0034262 (autophagy in response to cellular starvation) und GO:0042477 (odontogenesis of calcareous or chitinous tooth). In einem weiteren Schritt kann der Nutzer nun zu den Detailänderungen des entsprechenden Konzepts navigieren, was im nächsten Abschnitt detailliert erläutert wird.

7.4.2 Konzept-basierte Analyse

Der Workflow Konzept-basierte Evolutionsanalyse unterstützt die Untersuchung von Änderungen an Einzelkonzepten einer Ontologie, d. h. Nutzer können die Änderungshistorie von Konzepten einsehen. Dies ist insbesondere für Kuratoren von Annotationen sinnvoll, da sie nun die Möglichkeit besitzen die Historie eines Konzepts beispielsweise die Änderungen an Synonymen, Definitionen oder Namen nachzuvollziehen. Diese Informationen können sich während des manuellen Annotationsprozesses als hilfreich erweisen und zu einer Entscheidungsfindung beitragen. Der Workflow selbst startet mit einer Suche nach relevanten Konzepten. Dabei können Nutzer entweder direkt eine ihnen bekannte Konzept-ID (accession) eingeben oder mit Hilfe von Stichworten relevante Konzepte lokalisieren. Die Suche ist über alle Ontologien hinweg oder in ausgewählten Ontologien möglich. Wie in Abschnitt 7.4.1 gezeigt, kann dieser Workflow auch über die Quantitative Evolutionsanalyse erreicht werden, indem man ein geändertes Konzept innerhalb eines Versionswechsels weiterverfolgt. Bei einem selektierten Konzept werden dann im Workflow Informationen über den aktuellen Status und Inhalt angezeigt. Zusätzlich wird eine tabellarische Historie angeboten.

Das in Abb. 7.3 dargestellte Szenario illustriert eine konzept-basierte Analyse für Konzepte mit dem Namen „blood coagulation“ über alle verfügbaren Ontologien. Zunächst wird mit Hilfe des *Search Panel* eine String-basierte Suche nach dem Ausdruck „blood coagulation“ durchgeführt. Die gefundenen Ergebnisse, d. h. Konzepte in den Ontologien, welche „blood coagulation“ (Blutgerinnung) enthalten, werden innerhalb des Fensters *Search Results* präsentiert. Für das Beispiel existieren 7 Übereinstimmungen innerhalb von GO-BP und 7 weitere in GO-MF bzw. der MammalianPhenotypeOntology. Der Nutzer besitzt nun die Möglichkeit eines der ihm am interessantesten erscheinenden Konzepte aus der Ergebnisliste auszuwählen und dessen Evolution zu untersuchen. Wird beispielsweise das Konzept GO:0007596 – „blood coagulation“ aus GO-BP selektiert, erscheint ein weiteres Fenster (*Concept Evolution*), welches dessen Historie und Änderungen anzeigt. Das Fenster besteht aus zwei Teilen. Im ersten Teil werden die aktuellen Informationen, d. h. Attribute wie Name, Definition oder Synonyme und aus- bzw. eingehende Beziehungen aus

Abbildung 7.3: Workflow Konzept-basierte Analyse am Beispiel des Konzepts GO:0007596 – „blood coagulation“

der letzten Version präsentiert. Des Weiteren werden einige historische Kennzahlen wie das Erstellungsdatum oder Zeiträume von Abwesenheit angezeigt. Unter Abwesenheit wird hierbei das Fehlen eines Konzepts innerhalb einer Zeitperiode mit anschließender Wiedereinführung in die Ontologie verstanden. Im zweiten Teil wird die gesamte Historie des Konzepts tabellarisch präsentiert. Es werden der initiale Status des Konzepts, d. h. alle Attribute und Beziehungen zum Zeitpunkt der Erstellung, und alle Änderungen am Konzept wie Einfügungen, Modifikationen oder Löschungen von Attributwerten oder Beziehungen aufgelistet. Das Konzept GO:0007596 – „blood coagulation“ wurde in der Version vom Dezember 2002 in GO-BP eingeführt und war seither in allen Versionen verfügbar. Beziehungen zu Elternkonzepten wie GO:0007599 – „hemostatsis“, GO:0050817 – „coagulation“ und GO:0042060 – „wound healing“ wurden zwischen 2003 und 2005 eingefügt. Weitere Änderungen betrafen insbesondere Synonyme, beispielsweise wurde der Ausdruck „blood clotting“ temporär gelöscht und das Synonym „blood coagulation factor activity“ war nur zwischen 2005 und 2007 gültig.

7.4.3 Annotation-Migration

Der dritte Workflow *Annotation Migration* ermöglicht die Migration (Anpassung) von Annotationen an neuere Ontologieversionen. Dabei werden veraltete Annotationen ermittelt und können automatisch an die neue (aktuell gültige) Ontologieversion angepasst werden. Die dann gültigen Annotationen können beispielsweise für die Wiederholung eines Experiments oder einer Analyse basierend auf der neuen Ontologieversion genutzt werden. Die zu aktualisierenden Annotationen können über CSV Datenformate wie dem GOA Format [7] an die Applikation übermittelt werden. Der Workflow analysiert dann die Änderungen zwischen den in den Annotationen verwendeten Konzepten und denen der aktuellen Ontologieversion. Die abschließende Anpassung findet dann auf Basis der gefundenen Änderungen statt.

Der für das Beispiel in Abb. 7.4 verwendete Datensatz umfasst Proteinannotationen von Schweinen aus dem AgBase Projekt [75]. Die Annotationen basieren auf der Gene Ontology Version von Mai 2008, d. h. die Annotationen sollten auf eine aktuellere GO Version angepasst werden. Das Fenster *Migration Input Form* ermöglicht es dem Nutzer nötige Parameter für die Migration anzugeben. Dazu gehören das Datum der zur Annotation verwendeten Ontologieversion, das Datum der neuen Ontologieversion, Informationen zum verwendeten CSV Format sowie die Annotationen selbst. Die Annotationsdaten können einfach in das vorhandene Textfeld einfügt werden.

Nach Übermittlung der Daten an das System öffnet sich ein Fenster *Changed Concepts and Affected Annotations*, welches den Nutzer über die gefundenen Änderungen zwischen der verwendeten und neuen Ontologieversion informiert. Dabei werden drei Typen von Änderungen, welche einen Einfluss auf die Annotationen besitzen, unterschieden: Löschungen, das Zusammenfassen sowie das „Obsolete“-Markieren von

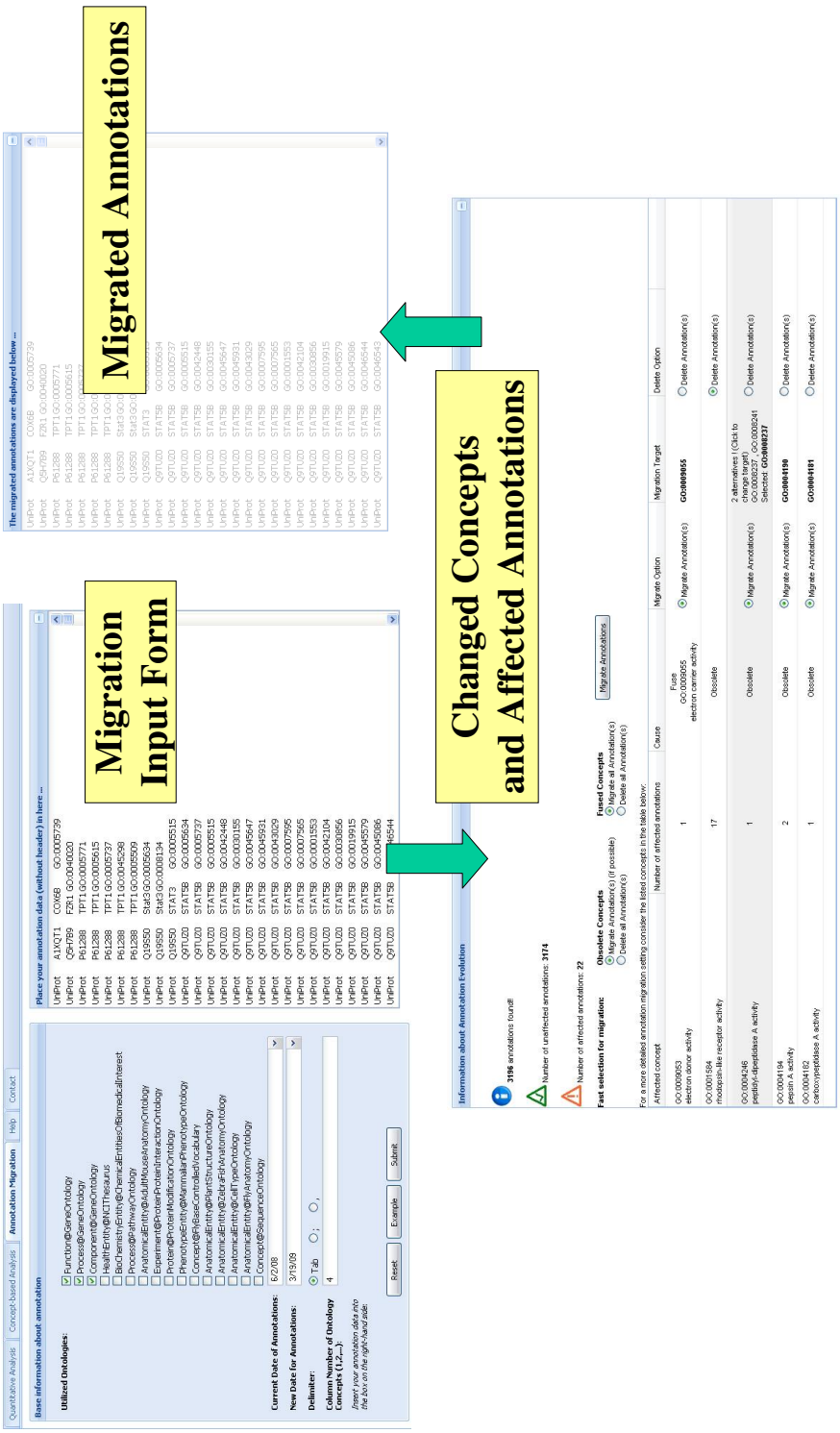


Abbildung 7.4: Workflow Annotation-Migration am Beispiel veralteter Genprodukt Annotationen bei Schweinen

Konzepten. Eine Tabelle listet alle identifizierten Änderungen sowie die zugehörige Anzahl betroffener Annotationen auf. An dieser Stelle besitzt der Nutzer nun diverse Möglichkeiten seine veralteten Annotationen an die neue Ontologieversion anzupassen. Im Falle von Konzeptlöschungen werden betroffene Annotationen ebenfalls entfernt. Annotationen, welche veraltete (obsolete) Konzepte verwenden, können entweder angepasst oder gelöscht werden. Da einige Ontologieprovider für veraltete Konzepte Informationen zu alternativen Konzepten anbieten, ist oftmals eine Anpassung der Annotationen anstatt einer direkten Löschung möglich. Neben den Ontologieversionen wurden zusätzlich Mappings zwischen veralteten und alternativen Konzepten ins Repository integriert. Beispielsweise stellt das GO Konsortium Vorschläge für Alternativkonzepte auf ihrer Webseite[32] bereit. In einigen OBO Ontologien sind Alternativen oftmals in der Konzeptbeschreibung oder in Kommentaren hinterlegt. Sind mehrere Anpassungsvarianten für eine Annotation verfügbar (z. B. bei multiplen Alternativen für ein veraltetes Konzept) kann der Nutzer manuell entscheiden, welche Variante verwendet werden soll. Für Konzepte, welche in einem anderen Konzept zusammengefasst wurden, ist eine Anpassung auf das zusammengefasste Konzept oder eine Löschung der Annotation möglich. Nutzer können die Migrationsoptionen für eine Änderungsart allgemein setzen (z. B. lösche alle Annotationen, welche veraltete Konzepte verwenden) oder individuelle Einstellungen vornehmen.

In den Annotationen des Beispiels von Mai 2008 existieren vier veraltete Konzepte sowie eine Zusammenfassung von Konzepten. Die Änderungen haben Einfluss auf insgesamt 22 Annotationen. Beispielsweise bestehen für das veraltete Konzept GO:0004246 – „peptidyl-dipeptidase A activity“ zwei Alternativen für die Migration GO:0008237 – „metallopeptidase activity“ und GO:0008241 – „peptidyl-dipeptidase activity“ zwischen welchen der Nutzer wählen kann. Abschließend werden die angegebenen Migrationsoptionen verwendet, um die veralteten Annotationen auf die neue Ontologieversion anzupassen. Das Fenster *Migrated Annotations* zeigt das Ergebnis der Migration, d. h. Nutzer können nun die aktualisierten und gültigen Annotationen kopieren und in ihren Analysen usw. verwenden.

7.5 Aktueller Stand

Zum Zeitpunkt September 2010 stellt OnEX 16 Ontologien der Lebenswissenschaften mit insgesamt ca. 700 Versionen für Evolutionsanalysen bereit. Die Ontologien umfassen die Gene Ontology und ihre Subontologien sowie Ontologien aus dem OBO Foundry. Das Versionierungsmodell gestattet eine effiziente Verwaltung großer Ontologien und zugehöriger Versionen. So liegen derzeit ca. 150.000 Konzepte, 260.000 Beziehungen und 1.000.000 Attribute aus verschiedenen Versionen im Repository

[32] `http://www.geneontology.org/doc/obsoletes-exact`, `http://www.geneontology.org/doc/obsoletes-inexact`

vor. Um eine unnötige Speicherung redundanter Informationen zu vermeiden, werden neben der ersten Version lediglich die Änderungen zu den nachfolgenden Versionen erfasst. Der Inhalt einer Ontologie zu einem konkreten Zeitpunkt (z. B. der Veröffentlichung einer Version) erfolgt über Zeitstempel. OnEX ist dabei nicht auf spezielle Ontologieformate limitiert und kann dadurch künftig weitere Ontologien anderer Formate integrieren.

Die Webapplikation enthält zusätzlich Hilfeseiten, welche den Einstieg für neue Nutzer erleichtern. So werden die Workflows schrittweise erläutert und Beispieldaten ermöglichen ein schnelles Kennenlernen der Applikation. OnEX ist seit 2008 online verfügbar und wurde bereits für zahlreiche Analysen mit verschiedenen Ontologien verwendet.

7.6 Zusammenfassung

Das in diesem Kapitel vorgestellte System OnEX ermöglicht einen benutzerfreundlichen und interaktiven Zugang zu Informationen über die Evolution und Änderungen in Ontologien der Lebenswissenschaften. Nutzer können Ontologien aus ihrem Interessengebiet bzgl. Evolution untersuchen, z. B. Änderungen an einer Ontologieversion einsehen, welche sie in einer Analyse oder Anwendung verwenden wollen. So ist u. a. eine Abschätzung des Einflusses von Ontologieänderungen auf Analyseergebnisse möglich. Außerdem können Anwender abschätzen, ob die Wiederholung einer Analyse auf einer neuen Version sinnvoll erscheint oder nicht. OnEX enthält aktuell 16 Ontologien mit 700 Versionen seit 2002 und ist unter `http://www.izbi.de/onex` verfügbar.

Die drei Workflows der Applikation adressieren diverse Aspekte der Evolutionsanalyse. Erstens, eine vergleichende quantitative Analyse der Evolution der Ontologien. Zweitens, die Untersuchung von Detailänderungen in der Historie von Einzelkonzepten einer Ontologie. Drittens, die semi-automatische Migration von Annotationen auf neuere Ontologieversionen.

8

Implementierungsaspekte – Speichereffiziente Versionierung großer Ontologien

8.1 Motivation

Typischerweise stellen Ontologieprovider vollständige Versionen einer Ontologie in Webarchiven oder auf Servern Nutzern zur Verfügung. Somit können konkrete Ontologieversionen geladen und anschließend innerhalb einer Ontologie-basierten Analyse oder Applikation verwendet werden. Dieses Vorgehen findet in den meisten wissenschaftlichen Arbeiten mit Ontologieverwendung statt, d. h. publizierte Ergebnisse einer Analyse oder eines Experiments beruhen in der Regel auf einer bestimmten Ontologieversion. Es findet somit eine Limitierung auf eine Ontologieversion statt, der Einfluss von Ontologieänderungen, beispielsweise gegenüber einer älteren Version oder der nächsten Version spielt keine Rolle. Für versionsübergreifende Analysen bzw. Experimente ist ein Zugang zu multiplen Versionen einer Ontologie nötig. Der Bedarf für eine Unterstützung der Archivierung wissenschaftlicher Daten inkl. dem Zugang zu vergangenen Dokumentversionen wurde bereits erkannt [17]. Es existieren auch bereits Systeme zur Archivierung wissenschaftlicher Daten (z. B. XArch [84] auf der Basis von XML). Leider bieten die Ontologieprovider lediglich einen Zugang zum Download einzelner Versionen an, eine Arbeit mit mehreren Versionen ist folglich nur mit viel Aufwand möglich. Die Bereitstellung eines einfachen Zugangs zu mehreren Versionen einer Ontologie erscheint somit für viele Applikationen im

Bereich der Ontologie-basierten Analysen sinnvoll. Beispielsweise gilt es oftmals zu prüfen, ob eine Analyse nochmals durchzuführen ist. Falls dies notwendig ist, muss die neue Ontologieversion erst in das Analysesystem integriert werden. So erfordern u. a. die in den Kapiteln 4–6 vorgestellten Algorithmen und Analysen sowie das OnEX System aus Kapitel 7 einen effizienten Zugang zu verschiedenen Versionen einer Ontologie. Dies betrifft insbesondere die Selektion relevanter Konzepte bzw. Strukturen innerhalb einer Ontologieversion. Zudem sollte aber auch ein einfacher, transparenter und versionsübergreifender Zugang zu Ontologieinformationen möglich sein.

Die zunehmende Größe der Ontologien in den Lebenswissenschaften (siehe Kapitel 4) erfordert eine effiziente Versionierung und Management der Ontologieversionen. Ein einfacher (naiver) Versionierungsansatz würde jede einzelne Version komplett mit ihrem gesamten Inhalt sichern. Dies würde jedoch bei der Größe der Ontologien sowie der häufigen Veröffentlichung neuer Versionen schnell zu einer erheblichen Datenmenge und damit zu Speicherengpässen führen. Da zwischen zwei Ontologieversionen ein Großteil der Elemente in der Ontologie unverändert bleibt (siehe Evaluierungsergebnisse in Kapitel 5), gilt es diese Redundanz bei der Versionierung zu vermeiden. Weiterhin muss die Heterogenität der Ontologieformate überwunden werden, damit Ontologien verschiedener Formate versioniert werden können. Eine Vielzahl der Ontologien in den Lebenswissenschaften werden im populären OBO Format veröffentlicht, allerdings existieren auch andere Formate wie CSV, OWL, RDF oder relationale Datenbanken, welche ebenfalls berücksichtigt werden müssen.

Der in diesem Kapitel vorgestellte Versionierungsansatz fokussiert auf die speichereffiziente Versionierung großer Ontologien. Die Beiträge umfassen:

- Es wird ein skalierbarer sowie speichereffizienter Ansatz zur Integration und zum Management vieler Versionen verschiedener Ontologien in einem zentralen Repository vorgeschlagen. Ein einheitlicher und transparenter Zugang zu integrierten Ontologien und ihren Versionen ermöglicht Applikationen ein versionsübergreifendes Arbeiten. Der Ansatz wurde erfolgreich für die Versionierung großer Ontologien in den Lebenswissenschaften verwendet, ist jedoch generisch und kann daher ebenfalls zur Versionierung von Ontologien anderer Domänen eingesetzt werden.

- Der Ansatz besteht aus einem Versionierungsmodell, welches die Basis für eine effiziente Ontologieversionierung darstellt. Es erlaubt die Erkennung von Änderungen an Konzepten, Beziehungen sowie Attributen über verschiedene Versionen hinweg. Im Gegensatz zu statischen Versionen einer Ontologie werden die Elemente einer Ontologie dynamisch auf Basis ihrer Lebenszeit verwaltet (versioniert).

- Auf Basis verschiedener Versionen der Gene Ontology wurde eine Evaluierung des Ansatzes mit Hinblick auf die Speichereffizienz durchgeführt. Zusätzlich

wurde die Ausführungszeit von Anfragen über den integrierten Ontologieversionen evaluiert.

Der Ansatz wurde in einem relationalen Repository umgesetzt und erfolgreich in der Praxis angewandt. Das Repository wird im Rahmen von Evolutionsanalysen [38, 116] sowie in anderen Projekten eingesetzt. Das in Kapitel 7 präsentierte OnEX System nutzt das Repository, um webbasierte Evolutionsanalysen an Ontologien zu ermöglichen. Derzeit sind ca. 700 Versionen von 16 verschiedenen Ontologien über OnEX zugreifbar. Des Weiteren wird das Versionierungssystem u. a. zur Analyse der Abhängigkeit von Ontologieänderungen auf die Ergebnisse von Analysen mit Hilfe des FUNC-Package [102] verwendet.

8.2 Versionierungsmodell

Das Versionierungsmodell basiert auf dem Ontologiemodell von Kapitel 3.1.2, d. h. es wird eine lineare Versionierung von Ontologieversionen $O_v = (C_v, A_v, R_v, t)$ unterstellt. In Ergänzung zum Ontologiemodell wird innerhalb des Versionierungsmodells jedem Element einer Ontologie eine Lebenszeit (t_{start}, t_{end}) zugeordnet. Diese Lebenszeit gibt an, in welchem Zeitraum ein Element gültig ist. So ist es möglich herauszufinden, ob ein Ontologieelement zu einem bestimmten Zeitpunkt t gültig ist (falls $t_{start} \leq t \leq t_{end}$) oder nicht. Die Lebenszeit eines Elements beginnt zum Zeitpunkt t_{start}. Der Wert von t_{start} entspricht dem Veröffentlichungsdatum der Ontologieversion in der das Element erstmalig eingeführt wurde. Analog gibt t_{end} das Ende der Lebenszeit eines Elements an. Der Wert von t_{end} ist der Zeitpunkt zu welchem ein Element letztmalig gültig war, d. h. das letzte Datum vor der Veröffentlichung einer neuen Version in der das Element nicht mehr vorhanden ist. Lebenszeiten können Konzepten, Beziehungen und Attributen zugeordnet werden, d. h. alle möglichen Elemente einer Ontologie werden dynamisch über die Lebenszeit versioniert. Will man eine bestimmte Ontologieversion O_v (gültig zum Zeitpunkt t) wiederherstellen, müssen alle Elemente, welche der Bedingung $t_{start} \leq t \leq t_{end}$ genügen, selektiert werden. Somit kann man flexibel den Stand einer Ontologie zu einem beliebigen Zeitpunkt rekonstruieren und die redundante Abspeicherung ganzer Ontologieversionen vermeiden.

Das soeben beschriebene Versionierungsmodell wurde innerhalb eines Repository (relationale Datenbank) umgesetzt. Das Repository erlaubt die Versionierung verschiedener Ontologien und ermöglicht einen einheitlichen Zugang zu den integrierten Versionen. Abb. 8.1 zeigt das relationale Schema des Repositories für das Versionierungsmodell. Das Schema besteht aus verschiedenen Entitäten zur konsistenten Repräsentation von Ontologien (Entität *Ontologies*), Ontologieversionen (Entität *Ontology Versions*), Konzepte (Entität *Ontology Concepts*) und die Ontologiestruktur (Entität *Ontology Relationships*). Die letzte Entität beschreibt die Beziehungen

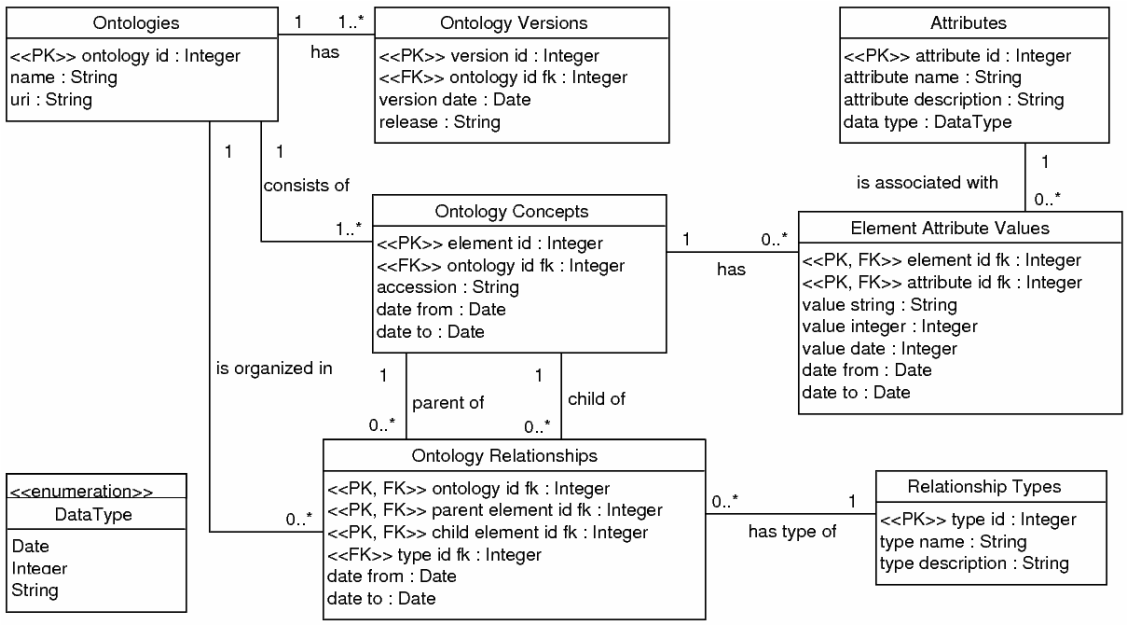


Abbildung 8.1: Relationales Schema des Versionierungsansatzes

innerhalb einer Ontologie, wobei zu jeder Beziehung dessen Typ (z. B. is_a, part_of) erfasst wird. Die flexible Verwaltung von Attributen beruht auf dem Entity-Value-Concept (EAV) [85]. Dabei beschreibt die Entität *Attributes* die Menge möglicher Attribute und ihre Semantik. Die eigentlichen Attributwerte werden in der Entität *Element Attribute Values* erfasst. Diese Trennung zwischen Attributsemantik und Attributwerten gestattet eine flexible Erweiterung ohne das relationale Schema anzupassen, z. B. wenn neue Attribute hinzukommen oder neue Attributwerte eingefügt werden müssen. Konzepte, Beziehungen und Attributwerte besitzen eine Lebenszeit, welche über die assoziierten Attribute *date-from* und *date-to* repräsentiert wird. Auf Basis dieser Lebenszeiten kann jede Ontologieversion dynamisch rekonstruiert werden. Über das Attribut *version-date* einer Ontologieversion ist lediglich ein Abgleich mit den Lebenszeiten der vorhandenen Elemente nötig (*date-from*$\leq$*version-date*$\leq$*date-to*), um die entsprechende Version zu rekonstruieren. Somit kann der Inhalt einer Ontologie für einen beliebigen Zeitpunkt nach der ersten verfügbaren Version bestimmt werden.

8.3 Integration und Änderungserkennung

Die häufige Veröffentlichung neuer Versionen einer Ontologie erfordert einen automatischen Mechanismus für deren Integration in das zentrale Repository. Dabei müssen Änderungen zwischen der zu integrierenden Version gegenüber der zuletzt aktuellen Version im Repository bestimmt werden. Auf Basis der Änderungen können dann

die Lebenszeiten der Elemente entsprechend angepasst oder neue Elemente integriert werden. Der Datenfluss bei der Integration einer neuen Ontologieversion besteht aus den folgenden drei Prozessen: (1) Laden und Import, (2) Änderungserkennung und (3) Anpassung der Elemente im Repository.

Die meisten Ontologieversionen können automatisch integriert werden. Dabei müssen der Typ und das Format der Ontologie berücksichtigt werden. Einige Ontologieprovider stellen Versionen ihrer Ontologie innerhalb eines eigenen webbasierten Versionierungssystems zur Verfügung (z. B. CVS oder SVN). Andere Provider hingegen legen die Versionen auf öffentlich verfügbaren Webseiten, Archiven oder FTP-Verzeichnissen ab. *Download Wrapper* für die verschiedenen Typen prüfen periodisch (z. B. jeden Monat), ob eine neue Version angeboten wird. Liegt eine neue Version vor, wird diese automatisch geladen. Die neue Version wird zunächst in einen temporären Bereich des Repositories importiert, anschließend findet eine Weiterverarbeitung statt. Format-spezifische *Importer* ermöglichen den Import verschiedener Ontologien indem Spezifika des jeweiligen Ontologieformats beachtet werden. Beispielsweise werden OWL bzw. RDF vorwiegend zur Beschreibung von Ontologien im Bereich des Semantic Web [56] verwendet. Viele Ontologien der Lebenswissenschaften liegen im OBO Format vor oder sind teilweise nur in proprietären Formaten basierend auf CSV, XML oder relationalen Datenbanken verfügbar. Die format-spezifischen Importer sowie die Wrapper zum Laden der Ontologieversionen werden flexibel miteinander kombiniert, um unnötige Implementierungen zu vermeiden. Während der Ladephase einer Ontologieversion werden deren Metadaten wie die Versionsnummer oder der Zeitpunkt der Veröffentlichung erfasst und ebenfalls im Repository hinterlegt. Diese Metadaten sind häufig nur in Dateinamen kodiert, z. B. bezeichnet „Thesaurus_10.08e.FLAT.zip“ die August 2010 Version des NCI Thesaurus, wobei mit 'e' die fünfte Version innerhalb des Monats August gekennzeichnet wird. In anderen Fällen liegen die Metadaten in speziellen Dokumentationsdateien vor oder können aus Systemen wie CVS oder SVN extrahiert werden.

Diese Metadaten bilden die Grundlage für die Änderungserkennung (Phase 2), in welcher die dynamischen Lebenszeitinformationen von Elementen aus den statischen Ontologieversionen gewonnen werden. Die Phase besteht aus den folgenden Schritten. Zunächst wird für die zu integrierende Version O_{v_i} die zugehörige Vorgängerversion $O_{v_{i-1}}$ im Repository ermittelt. Anschließend werden beide Versionen (O_{v_i} und $O_{v_{i-1}}$) miteinander verglichen, um hinzugefügte bzw. gelöschte Elemente zu bestimmen. Der Vergleich beruht auf den verfügbaren IDs (accessions) von Konzepten. Auf Basis der Änderungen können danach die Lebenszeiten der Elemente entsprechend angepasst werden. Neu hinzugekommene Elemente werden ebenfalls im Repository eingefügt. Ihre Lebenszeit erhält ein Startdatum (t_{start}), welches mit dem Datum der Veröffentlichung der importierten Version O_{v_i} übereinstimmt. Für gelöschte Elemente wird das Enddatum (t_{end}) auf den Zeitpunkt, zu welchem das Element letztmalig gültig war, gesetzt. Dies ist üblicherweise der Tag vor der Veröffentlichung der importierten Ontologieversion O_{v_i}. Die Anpassung der Lebenszeiten

Concepts		
accession number	*start*	*end*
GO:0009572	2002-02	**2007-05**
GO:0000446	**2007-06**	
...		

Relationships				
source	*target*	*type*	*start*	*end*
GO:0009572	GO:0044459	is_a	2006-05	**2007-05**
GO:0009572	GO:0009510	part_of	2003-05	**2007-05**
GO:0000446	**GO:0000347**	**is_a**	**2007-06**	
GO:0000446	**GO:0008023**	**is_a**	**2007-06**	

Attributes				
concept	*attribute*	*value*	*start*	*end*
GO:0009572	name	desmotubule central rod	2002-02	**2007-05**
GO:0009572	obsolete	false	2002-02	**2007-05**
GO:0000446	**name**	**nucleoplasmatic THO complex**	**2007-06**	
GO:0000446	**obsolete**	**false**	**2007-06**	
GO:0000446	**definition**	**The THO complex when ...**	**2007-06**	
GO:0009356	name	p-aminobenzoate synthetase complex	2002-12	**2007-05**
GO:0009356	**name**	**aminodeoxychorismate synthase complex**	**2007-06**	

Abbildung 8.2: Ausschnitt des Repositories für die Versionierung von GO Zellulären Komponenten

findet für Konzepte, Beziehungen sowie Attribute statt.

Das Beispiel in Abb. 8.2 illustriert einen Teil des Repositories während der Integration der Juni 2007 Version der GO Subontologie Zelluläre Komponenten. Die „jüngste“ Version im Repository stammt von Mai 2007. Somit muss die zu integrierende Version von Juni 2007 mit der Mai 2007 Version abgeglichen werden. Die fett markierten Einträge stellen Änderungen dar, welche durch die Integration der neuen Version entstanden sind. So wurde u. a. das Konzept GO:0009572 als gelöscht erkannt, wodurch das Enddatum dieses Konzepts sowie seiner Beziehungen und Attribute auf 2007-05 gesetzt wurden. Im Gegensatz dazu wurde z. B. das Konzept GO:0000446 mit den Attributen („name“, „obsolete status“, „definition“) sowie Beziehungen zu GO:0000347 und GO:0008023 neu eingefügt. Im Ergebnis wurde im Repository für jedes Element (Konzept, Beziehung, Attribut) ein neuer Eintrag mit Startdatum 2007-06 angelegt. Eine Attributwertänderung betraf das Konzept GO:0009356. Der Name des Konzepts wurde von „p-aminobenzoate synthetase complex“ auf „aminodeoxychorismate synthase complex“ abgeändert. Im Repository wird das Enddatum des alten Attributwertes auf 2007-05 gesetzt, somit ist der Attributwert nur bis Ende Mai 2007 als gültig anzusehen. Entsprechend wurde ein neuer Eintrag mit Startdatum 2007-06 eingefügt, welcher den ab Juni 2007 gültigen Konzeptnamen enthält.

8.4 Evaluierung

In diesem Abschnitt wird der vorgestellte Ansatz zur Versionierung großer Ontologien bzgl. Speichereffizienz und Anfrageperformanz evaluiert. Es werden zunächst die verwendeten Metriken für die Evaluierung eingeführt. Danach wird das Evaluierungsszenario beschrieben und die erzielten Ergebnisse präsentiert.

8.4.1 Evaluierungsmetriken

Für die Feststellung der Speicheranforderungen wird die Anzahl gespeicherter (versionierter) Elemente verwendet. Es wird zwischen dem vorgeschlagenen Versionierungsansatz (*approach*) und der redundanten Versionierung der Ontologieversionen (*naive*) unterschieden. Die folgenden Metriken zur Bestimmung der Anzahl von Elementen in einer Ontologieversion finden innerhalb der Evaluierung Anwendung:

- $|C|_{v_i}$, $|R|_{v_i}$, $|A|_{v_i}$: Anzahl von Konzepten, Beziehungen und Attributen in der Ontologieversion O_{v_i}
- $|E|_{v_i}$=$|C|_{v_i} + |R|_{v_i} + |A|_{v_i}$: Anzahl von Elementen in der Ontologieversion O_{v_i}

Die erste Version einer Ontologie wird mit v_1 und die letzte mit v_n gekennzeichnet. Die Gesamtanzahl aller Elemente bei der Integration von n Versionen $O_{v_1} \dots O_{v_n}$ einer Ontologie in ein Repository kann somit wie folgt bestimmt werden:

$$|E|_n^{naive} = \sum_{i=1}^{n} |E|_{v_i} \qquad |E|_n^{approach} = \left| \bigcup_{i=1}^{n} E_{v_i} \right|$$

Da der naive Versionierungsansatz jede Version komplett speichert, müssen die Anzahlen der Elemente in den einzelnen Version aufsummiert werden, um die Gesamtanzahl für n Versionen zu bestimmen. Im Gegensatz dazu vermeidet der vorgeschlagene Ansatz die Speicherung redundanter (unveränderter) Elemente indem lediglich die Änderungen zwischen den Versionen gespeichert werden müssen. Zur Bestimmung des Speicherbedarfs müssen in diesem Fall lediglich die Elemente in der Vereinigung aller n Versionen betrachtet werden, d. h. ein Element wird nur einmalig bei seinem erstmaligen Auftreten gezählt. Da beide Ansätze die erste Version vollständig speichern müssen und somit für die erste Version die gleiche Elementanzahl $|E|_{v_1}$ vorliegt, wird zusätzlich das Wachstum gegenüber der ersten Version evaluiert: $\frac{|E|_n^{naive}}{|E|_{v_1}}$ bzw. $\frac{|E|_n^{approach}}{|E|_{v_1}}$.

Zur Abschätzung des Einflusses von Änderungen an Ontologien auf die beiden Versionierungsansätze werden Einfügungen, Löschungen und Modifikationen an Ontologieelementen betrachtet. Mit *add*, *del* und *mod* wird die durchschnittliche Anzahl eingefügter, gelöschter bzw. modifizierter Elemente pro Versionswechsel bezeichnet. Auf Basis der Elementanzahl in der ersten Version $|E|_{v_1}$ kann die Gesamtanzahl von Elementen für n Versionen abgeschätzt werden:

$$|E|_n^{naive} = n|E|_{v_1} + \frac{n(n-1)}{2}(add - del) \qquad |E|_n^{approach} = |E|_{v_1} + (n-1)(add + mod)$$

Die Anzahl gespeicherter Elemente für den naiven Versionierungsansatz ist n-mal größer als die Elemente der ersten Version, korrigiert um die Elemente, welche während der Evolution eingefügt oder gelöscht wurden. Modifikationen an Elementen

(z. B. Änderungen an Attributwerten wie Umbenennungen) haben keinen Einfluss auf die Gesamtanzahl versionierter Elemente, da ein unverändertes Element wie auch ein modifiziertes Element abgespeichert werden. Im Gegensatz dazu speichert der vorgeschlagene Ansatz lediglich die Elemente der ersten Version vollständig ab, danach werden nur noch Änderungen (Einfügungen oder Modifikationen) gespeichert. Löschungen reduzieren bei diesem Ansatz nicht die Gesamtanzahl der Elemente, da die Versionierung über Zeitstempel nur das Enddatum eines gelöschten Elements ändert, d. h. ein Element wird physisch nicht aus dem Repository entfernt.

Da der vorgeschlagene Versionierungsansatz die Ontologieversionen nicht einzeln (separat) ablegt, müssen individuelle Versionen mit Hilfe der Lebenszeiten von Elementen rekonstruiert werden. Dies könnte insbesondere für große Ontologien im Vgl. zum naiven Ansatz einen langsamen Prozess darstellen. Ebenfalls könnten Interaktionen mit verschiedenen, integrierten Versionen aufgrund der dynamischen Versionierung zeitlich unterschiedlich ausfallen. Aus diesem Gründen wird in der Evaluierung neben der Speichereffizienz ebenfalls die Anfrageperformanz des Ansatzes untersucht. Zu diesem Zweck werden die Antwortzeiten für drei typische Anfragen an Ontologien betrachtet und anhand der integrierten Ontologieversionen evaluiert:

- $Q1$: Ermittle alle Details, d. h. Attribute, Vorgänger- bzw. Nachfolgerkonzepte für ein gegebenes Konzept
- $Q2$: Ermittle alle Geschwisterkonzepte eines gegebenen Konzepts
- $Q3$: Ermittle alle Konzepte innerhalb des Subgraphen eines gegebenen Konzepts

Für jede der Anfragen wird die mittlere Antwortzeit t_Q aus 10 Testläufen bestimmt. Des Weiteren wird die Anzahl der Ergebniselemente r_Q einer Anfrage Q sowie die Durchschnittsantwortzeit für ein Ergebniselement $\frac{t_Q}{r_Q}$ ausgewertet.

8.4.2 Evaluierungsszenario und Ergebnisse

Zunächst werden die Speicheranforderungen für eine synthetische Ontologie gezeigt. Anschließend erfolgt die Evaluierung an Realwelt-Daten. Tab. 8.1 vergleicht die Speicheranforderungen für beide Versionierungsansätze für eine Beispielontologie mit 1.000 Elementen in der ersten Version sowie für verschiedene Anzahlen ($n = 10, 20, 30$) von Versionen. Des Weiteren werden die folgenden Änderungsraten angenommen: $add = 20$, $del = 5$ und $mod = 10$ Änderungen pro Versionswechsel. Die Tabelle zeigt, dass der vorgeschlagene Ansatz dramatisch die Speicheranforderungen gegenüber dem naiven Ansatz reduzieren kann. Für n Versionen speichert er nahezu n-mal weniger Elemente, ohne dabei Information zu verlieren (kein Informationsverlust). Die Ergebnisse zeigen außerdem, dass der vorgeschlagene Ansatz desto

n	$\lvert E\rvert_n^{naive}$	$\lvert E\rvert_n^{approach}$	$\frac{\lvert E\rvert_n^{approach}}{\lvert E\rvert_n^{naive}}$
10	10,675	1,270	0.12
20	22,850	1,570	0.07
30	36,525	1,870	0.05

Tabelle 8.1: Speicheranforderungen für $|E|_{v_1} = 1000$, $add = 20$, $del = 5$, $mod = 10$

besser abschneidet, je mehr Versionen zu integrieren sind. Beispielsweise weist der Quotient aus $|E|_n^{approach}$ und $|E|_n^{naive}$ für 10 Versionen einen Wert von 0,12 auf (ca. 90% Einsparung). Er fällt für mehr Versionen weiter ab, z. B. 0,05 bei 30 Versionen, was einer Ersparnis von 95% entspricht. Der Grund hierfür liegt in der Tatsache begründet, dass der vorgeschlagene Versionierungsansatz nach Integration der ersten Version nur noch die Unterschiede (Änderungen) betrachtet und somit eine redundante Versionierung von Elementen vermeidet.

Für die Evaluierung an Realwelt-Daten wurde die Subontologie Biologische Prozesse der Gene Ontology verwendet (GO-BP). Die Subontologie GO-BP wird zusammen mit den beiden anderen Subontologien Molekulare Funktionen (GO-MF) sowie Zelluläre Komponenten (GO-CC) durch das Gene Ontology Konsortium verwaltet und gepflegt. Durch die starke Verwendung von GO in zahlreichen Projekten und Studien der Lebenswissenschaften wird täglich eine aktualisierte Version den Anwendern zur Verfügung gestellt[33]. In dieser Analyse wird maximal eine Ontologieversion von GO-BP pro Monat betrachtet. Bei mehr als einer Version pro Monat wurde die zuerst veröffentliche Version verwendet. Auf dieser Grundlage wurden zwischen Januar 2004 und Juni 2009 62 Versionen für die Evaluierung verwendet[34]. Die erste Version von Januar 2004 weist die geringste Anzahl an Elementen auf: 8.112 Konzepte, 12.456 Beziehungen und 25.268 Attribute. Die Elementanzahl verdreifachte sich bis zur letzten Version von Juni 2009: 17.104 Konzepte, 34.248 Beziehungen und 85.767 Attribute. Die nachfolgende Evaluierung fokussiert auf drei Aspekte. Zunächst werden die Speicheranforderungen bzgl. GO-BP für den vorgeschlagenen und den naiven Versionierungsansatz miteinander verglichen. Anschließend wird der Einfluss des Versionierungsintervalls (z. B. monatliche oder vierteljährliche Versionierung) untersucht. Abschließend werden Ergebnisse der Evaluierung der Anfrageperformanz präsentiert. Alle Analysen zur Evaluierung des Ansatzes nutzten eine MySQL Datenbank zur Integration der Ontologieversionen basierend auf dem relationalen Schema aus Abschnitt 8.2. Die physische Gesamtgröße der Datenbank für alle 62 Versionen von GO-BP beträgt 40 MB. Die Datenbank lief auf einer Linux-

[33] `http://archive.geneontology.org/termdb/latest/`

[34] Es ist anzumerken, dass für 4 Monate des Zeitraums keine Versionen aus dem Archiv `http://archive.geneontology.org/termdb/` geladen werden konnten.

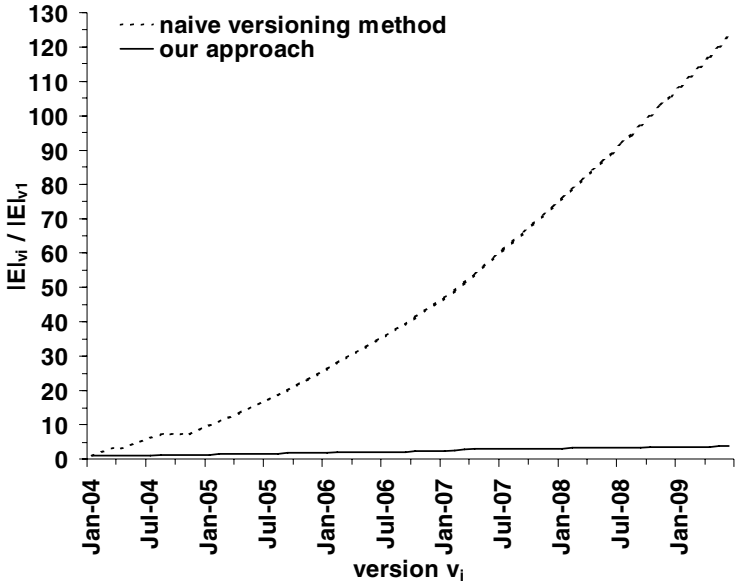


Abbildung 8.3: Vergleich des Speicherbedarfs für den vorgeschlagenen und naiven Versionierungsansatz

basierten 2X Dual-Core 2 GHz Maschine mit 8 GB physischen Arbeitsspeicher. Alle Analysen wurden mit deaktiviertem Cache im Einzelnutzermodus durchgeführt.

Abb. 8.3 zeigt die effizientere Speichernutzung des vorgeschlagenen gegenüber dem naiven Versionierungsansatz für die 62 integrierten Versionen im Zeitraum Januar 2004–Juni 2009. Die Speicheranforderungen (Anzahl integrierter Elemente $|E|_{v_i}$) werden für beide Ansätze normalisiert bzgl. der Elemente in der ersten Version ($\frac{|E|_{v_i}}{|E|_{v_1}}$) dargestellt. In beiden Ansätzen sind die Speicheranforderungen für die erste Version identisch (ca. 45.000 Elemente). Für den naiven Versionierungsansatz steigen die Speicheranforderungen sukzessive je mehr Versionen integriert werden. Nach Integration aller 62 Versionen wird im naiven Fall ein Speicherbedarf von 5,6 Millionen Elementen erreicht, was einem Wachstum von 124 gegenüber der ersten GO-BP Version entspricht. Im Gegensatz dazu besitzt der vorgeschlagene Versionierungsansatz einen Speicherbedarf von lediglich 170.000 Elementen für alle 62 Versionen. Dies entspricht einem moderaten Zuwachs von 3,8 gegenüber der ersten Version und korrespondiert mit dem tatsächlichen Wachstum der Ontologie von 45.000 auf 137.000 Elemente (Wachstum von ca. 3,0). Aufgrund der Tatsache, dass der naive Ansatz 32-mal mehr Speicherbedarf hat, würde die zugehörige Datenbank eine Größe von ca. 1,3 GB gegenüber der optimierten Größe von 40 MB besitzen.

In Abb. 8.4 werden die Speicheranforderungen für eine monatliche, viertel- bzw. halbjährliche Versionierung des vorgeschlagenen Ansatzes dargestellt. Natürlich sind i. Allg. die Speicheranforderungen für die monatliche Versionierung stets größer als die der beiden anderen. Jedoch fällt die Differenz zwischen den Varianten eher gering

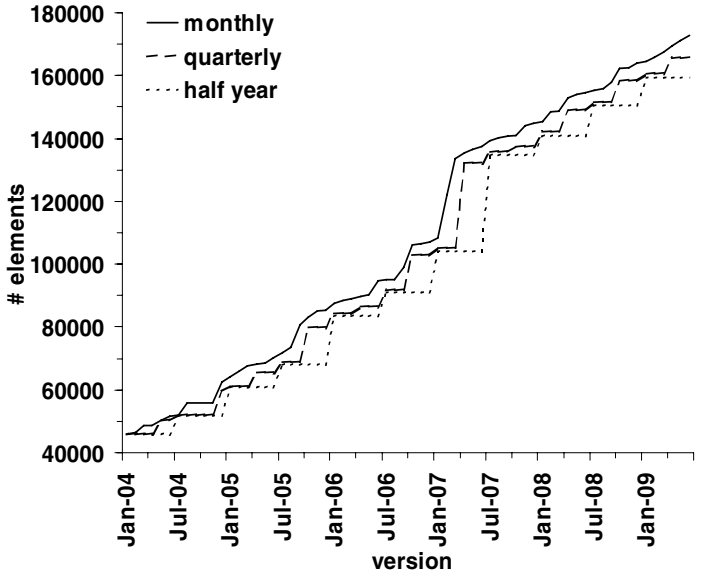


Abbildung 8.4: Vergleich des Speicherbedarfs für monatliche, viertel- bzw. halbjährliche Versionierung

aus. Zum Ende des betrachteten Zeitraums (Juni 2009) besitzt die monatliche Versionierung einen Speicherbedarf von ca. 170.000 Elementen verglichen mit ca. 165.000 bzw. 160.000 Elementen für die viertel- bzw. halbjährliche Versionierung. So können auch kürzere Versionierungsintervalle (z. B. auf Monatsbasis) mit geringfügig höheren Speicheranforderungen unterstützt werden. Dies unterstreicht die Skalierbarkeit des Ansatzes, so dass insbesondere auch große Ontologien mit häufigen Versionen effizient versioniert werden können.

In der letzten Analyse wird die Anfrageperformanz des Versionierungsansatzes mit Hilfe der drei Anfragen $Q1$, $Q2$ und $Q3$ untersucht. Für das Experiment wurden 22 Versionen zwischen Januar 2004 und April 2009 verwendet. Die Anfragen beziehen sich alle auf das Konzept „behavior“ (GO:0007610), welches auf der dritten Ebene der GO-BP Ontologie lokalisiert ist („biological process“ → „response to stimulus“ → „behavior“). Abb. 8.5 zeigt die Ausführungszeit pro Ergebniselement für die drei Anfragen $Q1$, $Q2$ und $Q3$. Die Anfragen weisen über alle Versionen eine nahezu konstante Ausführungszeit pro Ergebniselement auf. Es werden lediglich 0,17 ms ms zur Abfrage der Details ($Q1$) bzw. der Geschwister-Konzepte ($Q2$) eines Konzepts benötigt. Dahingegen benötigt die Abfrage des kompletten Subgraphen für GO:0007610 im Schnitt 1,47 ms pro Element. Die Ausführungszeiten verhalten sich über alle Versionen hinweg ähnlich, obwohl ein Zuwachs an Elementen in der Ontologie vorliegt (siehe vorherige Analysen). Abb. 8.6 zeigt die Ergebnisgrößen für alle drei Anfragen auf Basis des GO-BP Konzepts GO:0007610. Es ist zu erkennen, dass die Ergebnismengen in neueren Versionen anwachsen, d. h. das Wachstum der Gesamtontologie schlägt sich auch in größeren Ergebnismengen bei den drei Anfragen

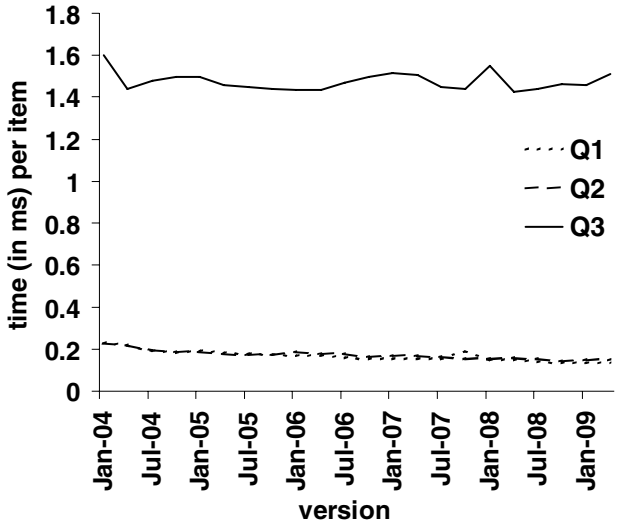


Abbildung 8.5: Anfrageperformanz-Analyse von $Q1$, $Q2$ und $Q3$ für GO:0007610

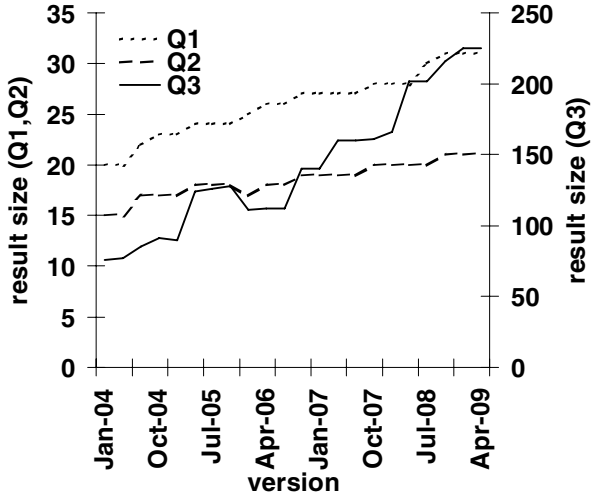


Abbildung 8.6: Ergebnisgrößen der Anfragen $Q1$, $Q2$ und $Q3$ für GO:0007610

nieder. So steigt die Anzahl von Konzeptdetails ($Q1$) von 20 auf 31 zwischen Januar 2004 und April 2009 (Wachstum von ca. 1,55). Für $Q2$ (Geschwisterkonzepte) und $Q3$ (kompletter Subgraph) liegt ein Zuwachs von 1,4 bzw. 3 für die Ergebnismengen vor. Zusammengefasst zeigen die Analysen, dass der vorgeschlagene Ansatz nicht nur speichereffizient große Ontologien versionieren kann, sondern auch gleichzeitig nahezu konstante Antwortzeiten für die betrachteten Anfragen ermöglicht.

8.5 Zusammenfassung

In diesem Kapitel wurde ein skalierbarer und speichereffizienter Ansatz zur Versionierung großer Ontologien vorgeschlagen. Das zugrunde liegende Versionierungsmodell verwaltet Lebenszeiten für alle Ontologieelemente (Konzepte, Beziehungen und Attribute). Dies ermöglicht die Rekonstruktion einer Ontologie zu einem beliebigen Zeitpunkt, ohne die einzelnen Ontologieversionen separat (einzeln) abzuspeichern. Der Ansatz wurde an Ontologien der Lebenswissenschaften erprobt (siehe Kapitel 7 zu OnEX), kann aber auch in anderen Domänen eingesetzt werden. Die Evaluierung des Ansatzes zeigte dessen hohe Speichereffizienz mit einer bis zu n-fachen Reduktion des Speicherbedarfs gegenüber der separaten Speicherung von n Versionen. Die Anfrageperformanz wurde über mehreren integrierten Ontologieversionen evaluiert und zeigte ebenfalls sehr gute Ergebnisse. Es konnte insbesondere gezeigt werden, dass die angewandte speichereffiziente Versionierung keinen signifikanten Einfluss auf die Antwortzeiten von Anfragen hat.

Teil IV

Zusammenfassung und Ausblick

9 Zusammenfassung und Ausblick

9.1 Zusammenfassung

Die vorliegende Arbeit beschäftigte sich mit dem Problem der Evolution von Ontologien im Bereich der Lebenswissenschaften. Im ersten Teil der Arbeit wurden dazu Algorithmen zur Analyse der Evolution und Änderungsbestimmung präsentiert. Der zweite Teil der Arbeit thematisierte zugehörige Systeme, welche Nutzern Evolutionsanalysen für Ontologien der Lebenswissenschaften ermöglichen.

9.1.1 Algorithmen zur Änderungsbestimmung

Zunächst wurde ein generelles Framework zur quantitativen Evolutionsanalyse für Ontologien und Mappings präsentiert. Das Framework wurde für eine umfassende Analyse der Evolution von 16 Ontologien der Lebenswissenschaften zwischen 2004 und 2008 verwendet. Die Analyse zeigte, dass alle untersuchten Ontologien stetig verändert (angepasst) werden und ein signifikantes Wachstum aufweisen. Die häufigste Art von Änderungen sind Einfügungen, jedoch wurden auch zahlreiche Konzepte gelöscht oder auf veraltet („obsolete“) gesetzt. Ein ebenfalls signifikantes Wachstum konnte für Instanzdaten sowie Ontologie-basierte Mappings (Annotation-Mappings und Ontologie-Mappings) festgestellt werden. Die Erkenntnisse aus den durchgeführten Analysen bildeten die Basis für die weiteren Arbeiten.

Erstens wurde ein neuartiger auf Regeln basierter Ansatz zur Bestimmung ausdrucksstarker und invertierbarer Diff Evolution-Mappings zwischen zwei Ontolo-

gieversionen entwickelt. Ein Evolution-Mapping deckt dabei sowohl einfache als auch komplexe Änderungen ab. Der Ansatz basiert auf einem Match-Mapping zwischen den Ontologieversionen und verwendet Regeln zur Erkennung einfacher sowie komplexer Änderungsoperationen. Es konnte theoretisch wie praktisch gezeigt werden, dass vollständige Evolution-Mappings sowie deren Inverse eine korrekte Migration von Ontologieversionen ermöglichen. Die semantisch ausdrucksstarken Diff Evolution-Mappings bilden zudem die Grundlage für die korrekte Anpassung (Migration) abhängiger Instanzen und Mappings. Gegenüber bisherigen Lösungen zeichnet sich der Ansatz insbesondere durch seine Flexibilität aus und kann damit den Anforderungen verschiedener Anwendungen gerecht werden. So kann die Menge relevanter Änderungsoperationen nach Bedarf verändert und ergänzt werden. Außerdem besteht die Möglichkeit die zugehörigen Regeln an die Gegebenheiten verschiedener Ontologien anzupassen, was eine flexible Änderungsbestimmung ermöglicht.

Ein zweiter Ansatz thematisierte das bisher noch nicht betrachtete Problem der Bestimmung änderungsintensiver bzw. stabiler Regionen innerhalb einer Ontologie. Dazu wurden die Notation von Ontologieregionen und zugehörige Metriken zur Beurteilung ihrer Änderungsintensität (Stabilität) eingeführt. Der vorgestellte automatisierte Algorithmus erlaubt die Bestimmung (in)stabiler Ontologieregionen auf Basis veröffentlichter Ontologieversionen in einem vorgegebenen Zeitraum. Dabei werden die Änderungen zwischen den Versionen betrachtet und mit Hilfe der Ontologiestruktur änderungsintensive bzw. stabile Ontologieregionen bestimmt. Somit können Ontologiedesigner die Entwicklung ihrer Ontologie beobachten und beurteilen, um beispielsweise künftige Arbeiten und Änderungen in Teilen der Ontologie zu planen und zu koordinieren. Ebenso können die Ergebnisse in Entscheidungsprozessen Anwendung finden. So können beispielsweise Endnutzer von Ontologien a-priori abschätzen, ob eine Neuberechnung ihrer Ontologie-basierten Analysen oder Experimente auf Basis einer neueren Ontologieversion als sinnvoll erscheint oder nicht. Eine vergleichende Evaluierung anhand zweier großer Ontologien der Lebenswissenschaften zeigte, dass der Algorithmus in der Lage ist instabile sowie stabile Ontologieregionen automatisiert zu klassifizieren.

9.1.2 Systeme zur Analyse der Ontologieevolution

Die webbasierte Applikation OnEX (Ontology Evolution Explorer) ermöglicht einen benutzerfreundlichen und interaktiven Zugang zu Informationen über die Evolution und Änderungen in Ontologien der Lebenswissenschaften. Nutzer können Ontologien aus ihrem Interessengebiet bzgl. Evolution untersuchen, indem sie beispielsweise Änderungen an einer Ontologieversion einsehen, welche in einer Analyse oder Anwendung genutzt werden soll. Somit ist beispielsweise eine Abschätzung des Einflusses von Ontologieänderungen auf Analyseergebnisse möglich oder Kuratoren können Informationen aus der Historie von Konzepten in ihre Entscheidungsfindung mit einbeziehen. OnEX enthält aktuell 16 Ontologien mit ca. 700 Versionen seit

2002 und ist unter `http://www.izbi.de/onex` verfügbar. Die drei Workflows von OnEX adressieren verschiedene Nutzerszenarios der Evolutionsanalyse: (1) eine vergleichende quantitative Analyse der Evolution der Ontologien, (2) die Inspektion von Detailänderungen in der Historie von Einzelkonzepten einer Ontologie und (3) die semi-automatische Migration von Annotationen auf neuere Ontologieversionen.

Für OnEX wurde ein skalierbarer und speichereffizienter Ansatz zur Versionierung großer Ontologien vorgeschlagen. Das Versionierungsmodell verwaltet dynamisch alle Elemente einer Ontologie (Konzepte, Beziehungen und Attribute) über sogenannte Lebenszeiten (Zeitstempel). Dies ermöglicht die Rekonstruktion einer Ontologie zu einem beliebigen Zeitpunkt ohne die einzelnen Ontologieversionen separat (redundant) abzulegen. Die Evaluierung des Ansatzes zeigte dessen hohe Speichereffizienz, da die redundante Speicherung unveränderter Elemente vermieden wird. Es konnte eine bis zu n-fache Reduktion des Speicherbedarfs gegenüber der separaten Speicherung von n Versionen erreicht werden. Die Anfrageperformanz zeigte ebenfalls sehr gute Ergebnisse, d. h. die angewandte speichereffiziente Versionierung hat keinen signifikanten Einfluss auf die Ausführungszeit von Anfragen.

9.2 Ausblick

Die in dieser Arbeit vorgestellten Ansätze und Algorithmen mit Fokus auf Ontologien der Lebenswissenschaften können aufgrund ihrer Flexibilität als Grundlage für erweiterte Ansätze bzw. Systeme dienen. Die Ansätze sind generisch angelegt und ermöglichen somit ebenfalls einen Einsatz in anderen Domänen wie beispielsweise dem Web oder E-Commerce. Zudem können aus den erzielten Analyseergebnissen Ideen für weitere Algorithmen abgeleitet werden. Nachfolgend werden kurz einige Richtungen für künftige Arbeiten dargestellt.

Stabilität Ontologie-basierter Mappings: Die Analyseergebnisse aus Kapitel 4 zeigten insbesondere hohe Instabilitäten in Ontologie-basierten Mappings wie Annotation- oder Ontologie-Mappings. Gerade automatisch generierte Mappings (z. B. durch das Matching von Ontologien oder automatische Annotationsverfahren) sind von Änderungen in den Ontologien bzw. Instanzen betroffen, d. h. Instabilitäten in den Einzelquellen werden in der Regel 1:1 auf abhängige Mappings übertragen. Applikationen oder Analysen, welche auf diesen Mappings basieren, müssen dann gegebenenfalls mit den vorhandenen Instabilitäten umgehen können. In solchen Fällen könnten Algorithmen die Stabilitäten der Mappings und ihrer Korrespondenzen einschätzen und entsprechend stabile und instabile Korrespondenzen filtern/klassifizieren. Eine erste Arbeit in dieser Richtung schlägt Metriken für die Bestimmung stabiler Korrespondenzen in automatisch erzeugten Ontologie-Mappings vor [116]. Eine weitere Arbeit untersuchte auf Basis der hier vorgestellten Erkenntnis-

se die Evolution von GO-basierten Annotation-Mappings in einem größerem Umfang. Erste vorgeschlagene Metriken nutzen historische Informationen (d. h. Informationen aus älteren Versionen) zur Selektion qualitativ hochwertiger Annotationen in einer aktuellen Version [38]. Jedoch bedarf es künftig weiterer Untersuchungen und Forschung, um Abhängigkeiten noch detaillierter zu erfassen und zu verstehen, z. B. der Einfluss automatischer Match-Verfahren auf die Stabilität der erzeugten Ontologie-Mappings.

Effiziente Adaption Ontologie-basierter Mappings unter Evolution: Wie im Kapitel zu Verwandten Arbeiten festgestellt wurde, existieren derzeit kaum Ansätze die eine Propagierung von Ontologieänderungen in abhängige Mappings ausreichend unterstützen. Liegt beispielsweise zwischen zwei Ontologien ein bereits akzeptiertes Ontologie-Mapping vor kann dieses für die Bestimmung von Ontologie-Mappings zwischen neueren Versionen der Ontologien wiederverwendet werden (Reuse). Wird für eine oder beide Ontologien eine neue Version veröffentlicht, müsste das nun veraltete Ontologie-Mapping angepasst werden, um weiterhin konsistent und aktuell zu sein. Der im Kapitel 5 vorgestellte Diff-Algorithmus könnte zunächst ein Diff Evolution-Mapping zwischen der alten und neuen Version einer Ontologie bestimmen. Anschließend kann auf Basis der erkannten Änderungen eine Adaption des veralteten Ontologie-Mappings stattfinden. So könnten u. a. unveränderte Teile direkt übernommen (wiederverwendet) werden. Hingegen erfordern von Änderungen betroffene Konzepte eine entsprechende Anpassung des Mappings. Neue hinzugekommenes Wissen in den Ontologien wie z. B. neu eingefügte Subgraphen sollten zur Wahrung der Aktualität ebenfalls im adaptierten Mapping abgedeckt werden.

Einfluss von Evolution auf Analyseergebnisse: Gerade in den Lebenswissenschaften werden Ontologien zur Interpretation von Analyseergebnissen aus Experimenten oder innerhalb statistischer Tests genutzt. Beispiele für derartige Analysewerkzeuge sind u. a. FUNC [102], GoMiner [125] oder GOTree Machine [126]. Die Ergebnisse basieren auf einer konkreten Version der verwendeten Ontologie. Dabei finden Änderungen zwischen Ontologieversionen keine Beachtung, obwohl Änderungen in den Ontologien höchstwahrscheinlich einen Einfluss auf die erzeugten Ergebnisse haben. Mit Hilfe des Versionierungsansatzes und den Algorithmen zur Bestimmung des Diff bzw. (in)stabiler Ontologieregionen wäre die Grundlage für eine mögliche Abschätzung des Einflusses gegeben. Analyseergebnisse, welche sich auf eine bestimmte Ontologieregion beziehen, sollten im Falle einer Instabilität dieser Region wiederholt werden. Liegt hingegen eine stabile Region vor, können die bisher erzielten Ergebnisse als gültig erachtet und übernommen werden. Jedoch müsste neben der Ontologieevolution auch die Evolution weiterer an den Analysen beteiligter Quellen und Mappings, z. B. Annotation-Mappings, untersucht und deren Einfluss abgeschätzt werden.

Instanz-basierte Bestimmung interessanter Ontologieregionen: Der in dieser Arbeit vorgestellte Algorithmus zur Bestimmung änderungsintensiver Ontologieregionen bezieht „nur“ Änderungen an der Ontologie selbst ein. Darüber hinaus könnten auch Änderungen in Ontologie-basierten Mappings oder Instanzen mit in die Bestimmung von Regionen einfließen. Gerade Veränderungen in Annotationen geben einen Hinweis darauf, welche Regionen aktuell sehr stark im Fokus stehen oder eher weniger Verwendung finden. Beispielsweise werden Publikationen in PubMed[35] mit Konzepten des MeSH [72] inhaltlich kategorisiert. Ein angepasster Algorithmus würde die Informationen über veränderte Annotationen nutzen, um aktuelle bzw. bereits tiefgründig untersuchte Forschungsthemen in den Lebenswissenschaften zu lokalisieren. In Wikipedia wäre ein ähnliches Vorgehen denkbar. Die vorhandenen bzw. neu erstellten Artikel werden je nach Thema und Inhalt bestimmten Kategorien zugeordnet. Auch hier finden ständige Anpassungen und Veränderungen statt, welche einen Hinweis auf aktuelle Themen geben. Zudem wären die Informationen für die Koordinierung der Entwicklung des Kategoriensystems der Wikipedia hilfreich.

Die Ausführungen zu möglichen künftigen Arbeiten zeigen, dass noch weitere offene Probleme existieren. Mit dieser Arbeit wurden erste Beiträge in Richtung der Evolution von Datenquellen und Mappings in den Lebenswissenschaften geleistet. Die generischen Algorithmen und Ansätze können aufgrund ihrer Flexibilität als Grundlage zur Lösung weiterer (noch ausstehender) Probleme Verwendung finden. Darüber hinaus ist eine Übertragung in andere Domänen als vielversprechend anzusehen.

[35] `http://www.ncbi.nlm.nih.gov/pubmed`

Teil V

Anhang

A

Evolution-Trendcharts für ausgewählte Ontologien

A.1 Vergleichende Trendcharts

Dieser Abschnitt fasst alle vergleichenden Trendcharts der quantitativen Evolutionsanalyse von Ontologien aus Kapitel 4 zusammen. Jedes Trendchart stellt den Verlauf der Anzahl der Konzepte über die analysierten Versionen dar. Zunächst wird ein Vergleich zwischen den drei Subontologien der GO präsentiert. Anschließend erfolgt ein Vergleich zwischen GO und NCI Thesaurus. Die vier anderen Trendcharts zeigen vergleichend die Evolution in den anderen Ontologien basierend auf deren Größe: (1) 10.000≤ $|C|$ ≤20.000, (2) 4.000≤ $|C|$ ≤10.000, (3) 1.000≤ $|C|$ ≤4.000 und (4) $|C|$ ≤1.000 .

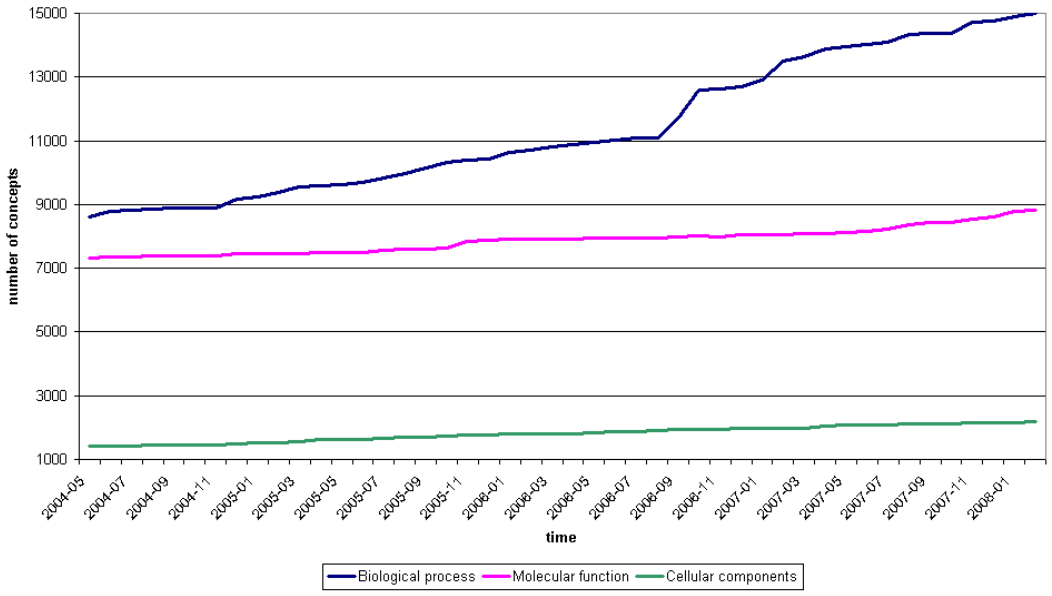


Abbildung A.1: Evolution der drei GO Subontologien

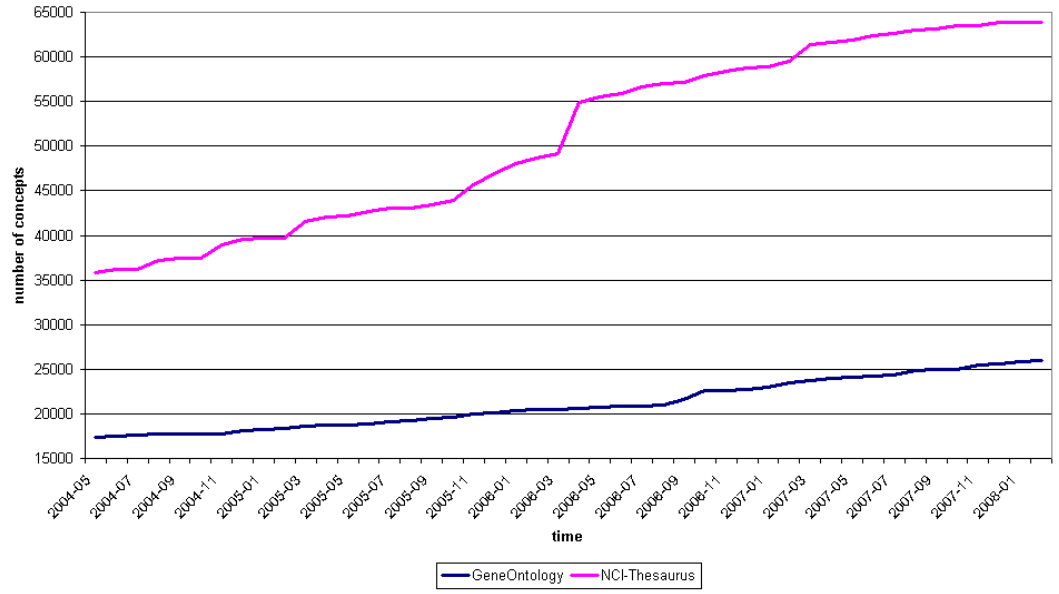


Abbildung A.2: Evolution von Gene Ontology und NCI Thesaurus

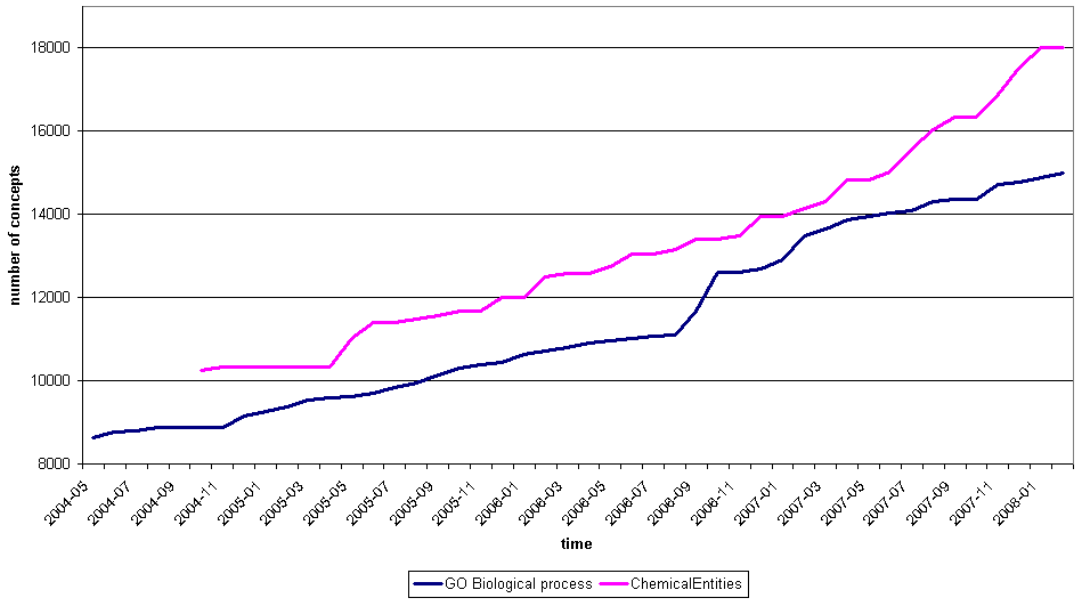


Abbildung A.3: Evolution von Ontologien mit $10\,000 < |C| < 20\,000$

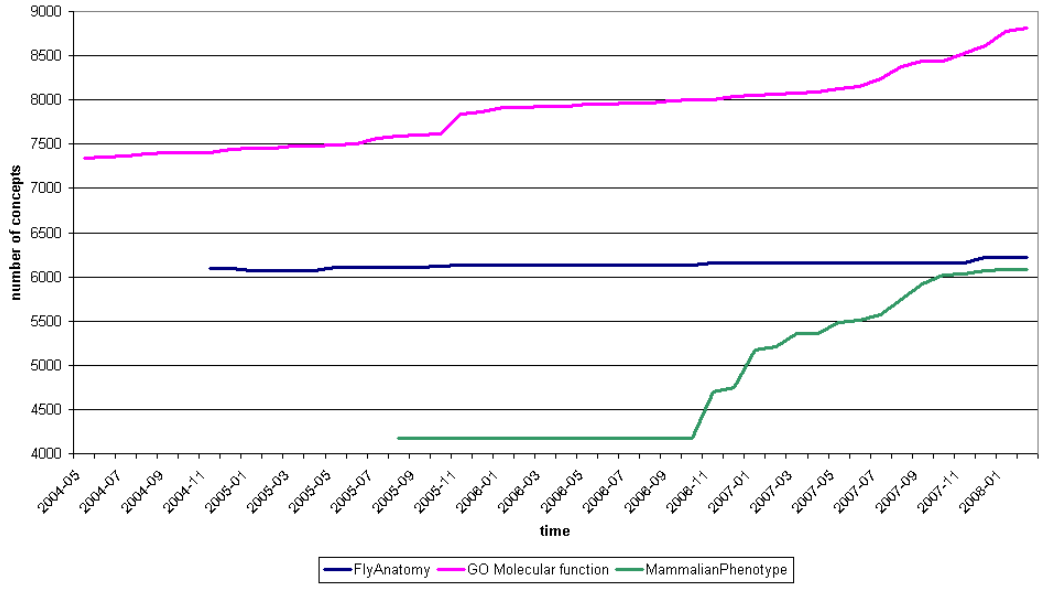


Abbildung A.4: Evolution von Ontologien mit $4.000 < |C| < 20.000$

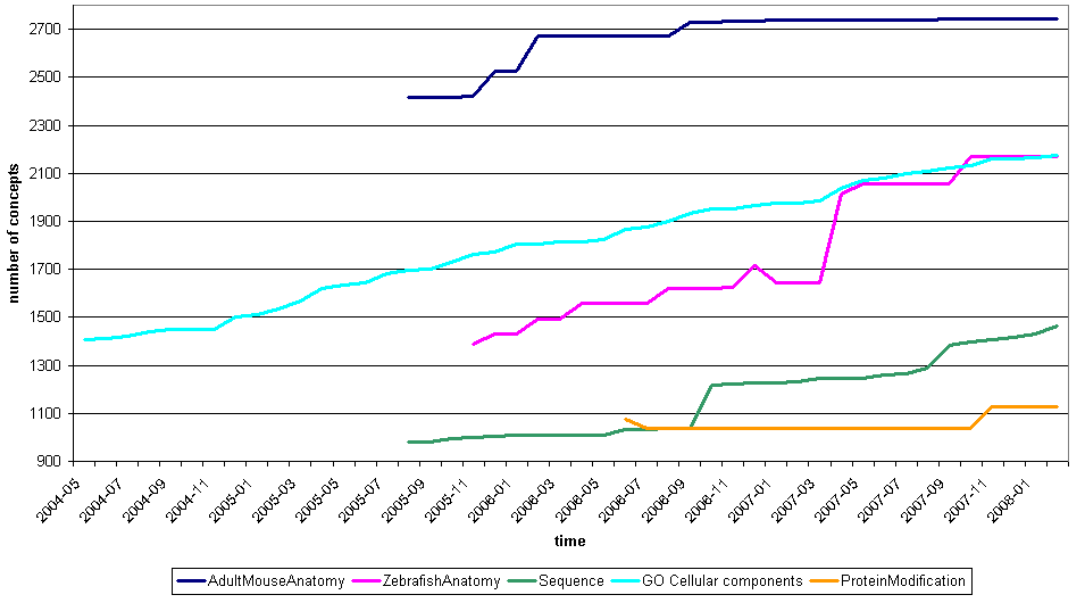


Abbildung A.5: Evolution von Ontologien mit $1.000 < |C| < 4.000$

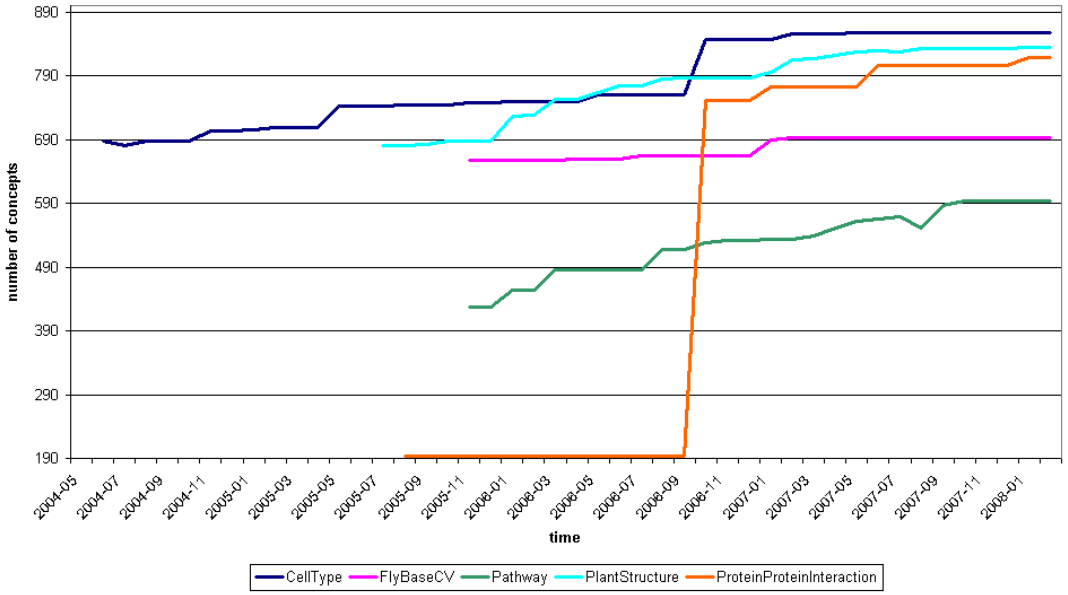


Abbildung A.6: Evolution von Ontologien mit $|C|<1.000$

A.2 Einzelne Trendcharts

Die nachfolgenden Trendcharts zeigen die Evolution jeder untersuchten Ontologie. In jedem Diagramm wird das Wachstum in der Anzahl der Konzepte gezeigt. Des Weiteren wird die Anzahl der Änderungen gruppiert nach *add* bzw. *del*/*toObs* pro Versionswechsel dargestellt.

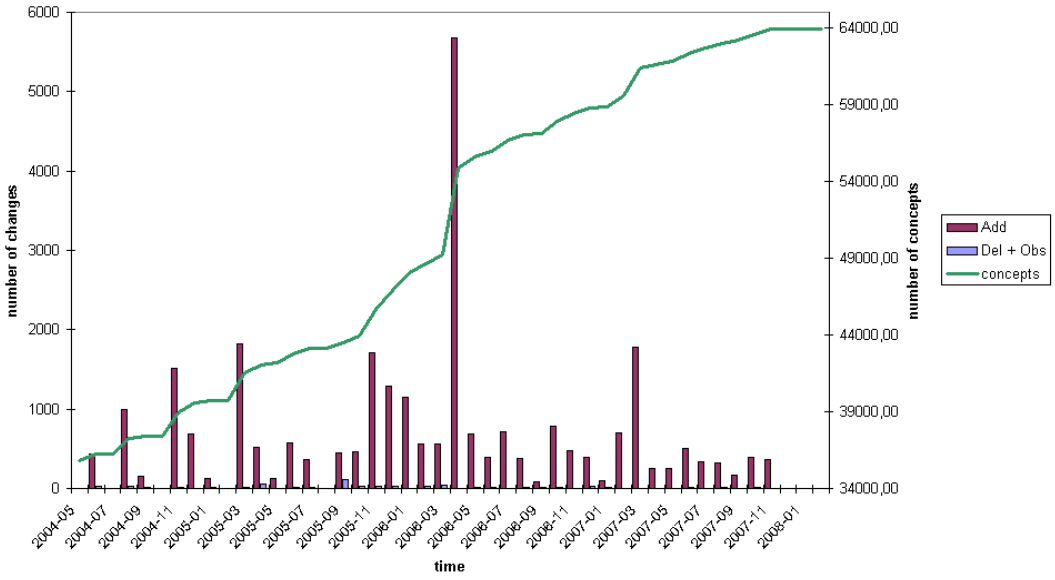


Abbildung A.7: Evolution von NCI Thesaurus

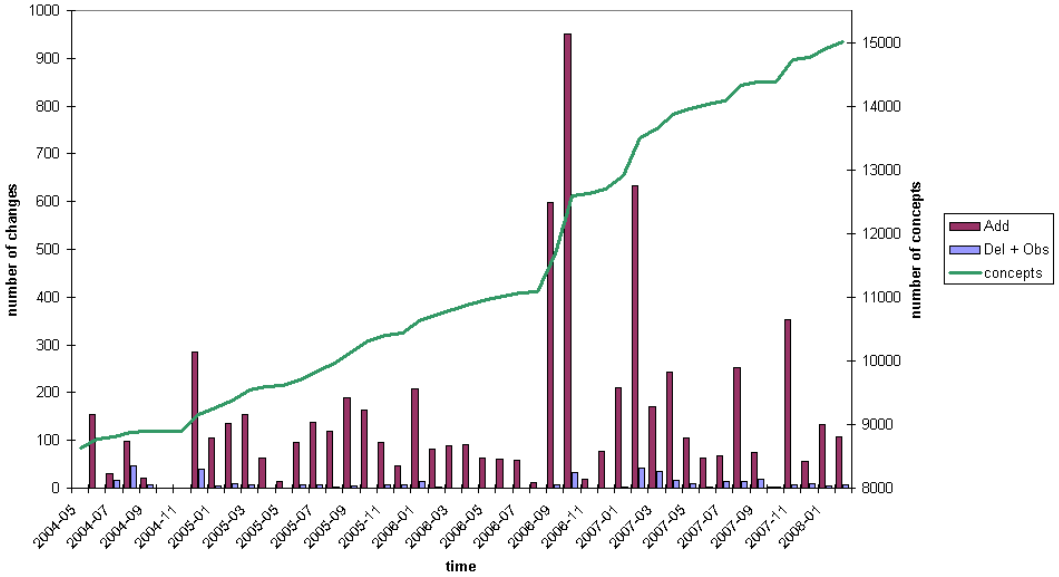


Abbildung A.8: Evolution der GO Subontologie Biologische Prozesse

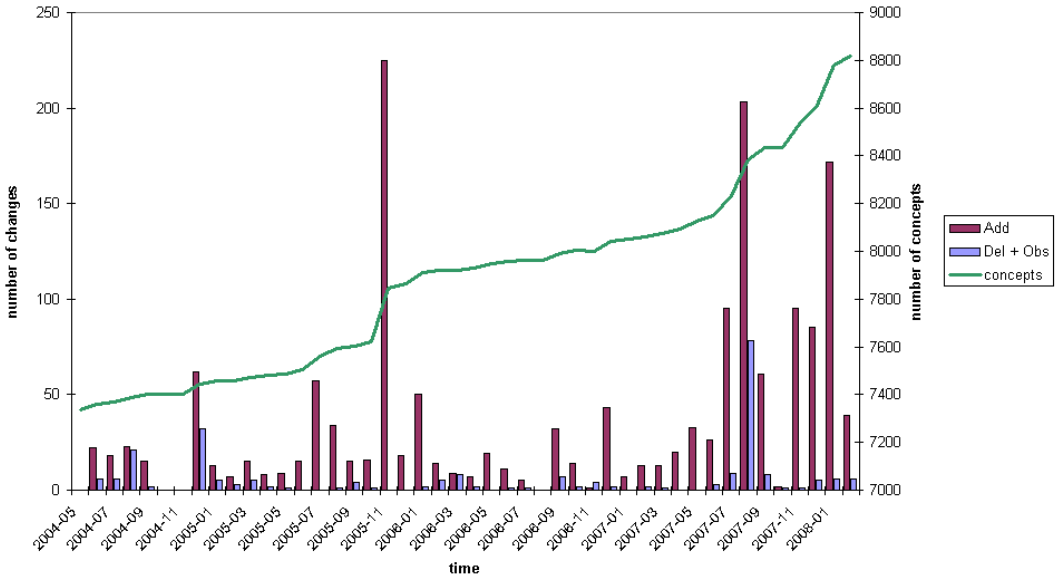


Abbildung A.9: Evolution der GO Subontologie Molekulare Funktionen

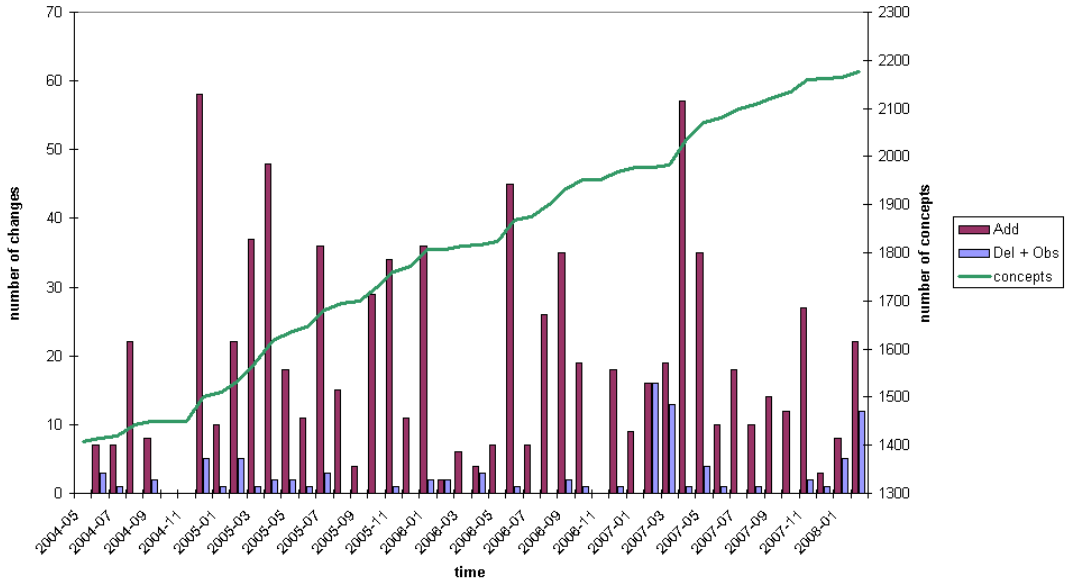


Abbildung A.10: Evolution der GO Subontologie Zelluläre Komponenten

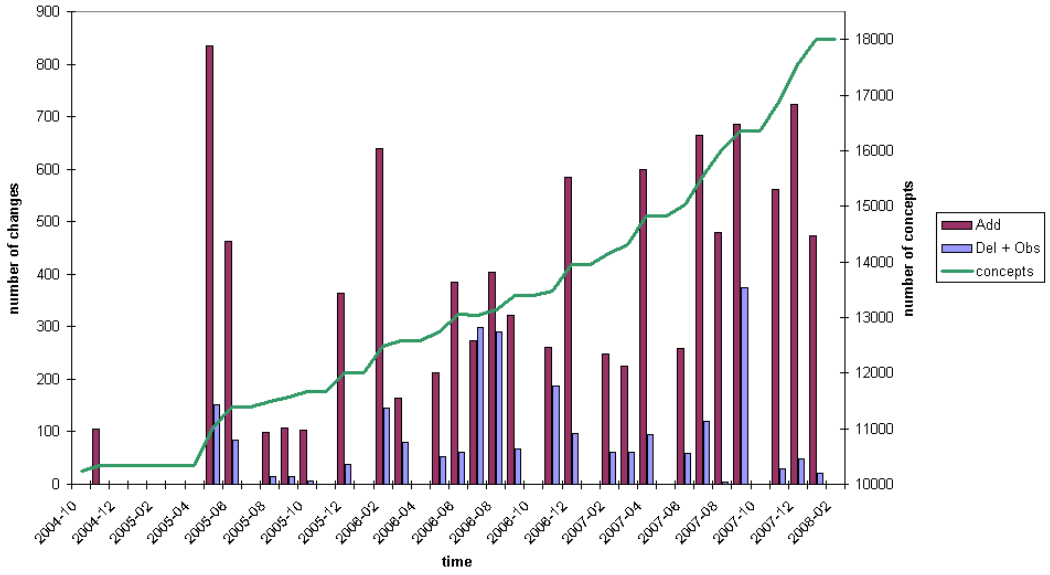


Abbildung A.11: Evolution von Chemical Enities of Biomedical Interest

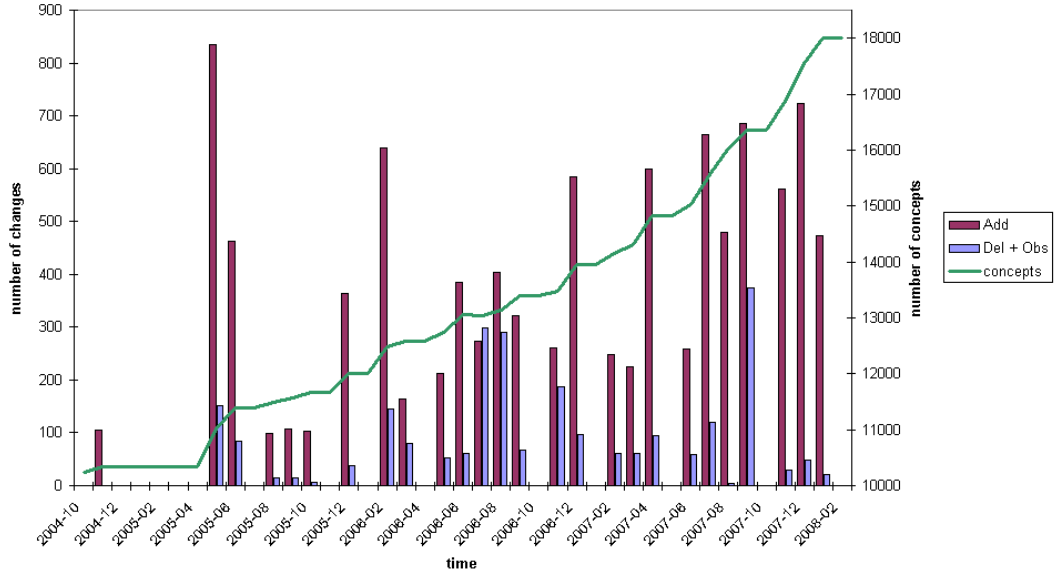


Abbildung A.12: Evolution von Fly Anatomy

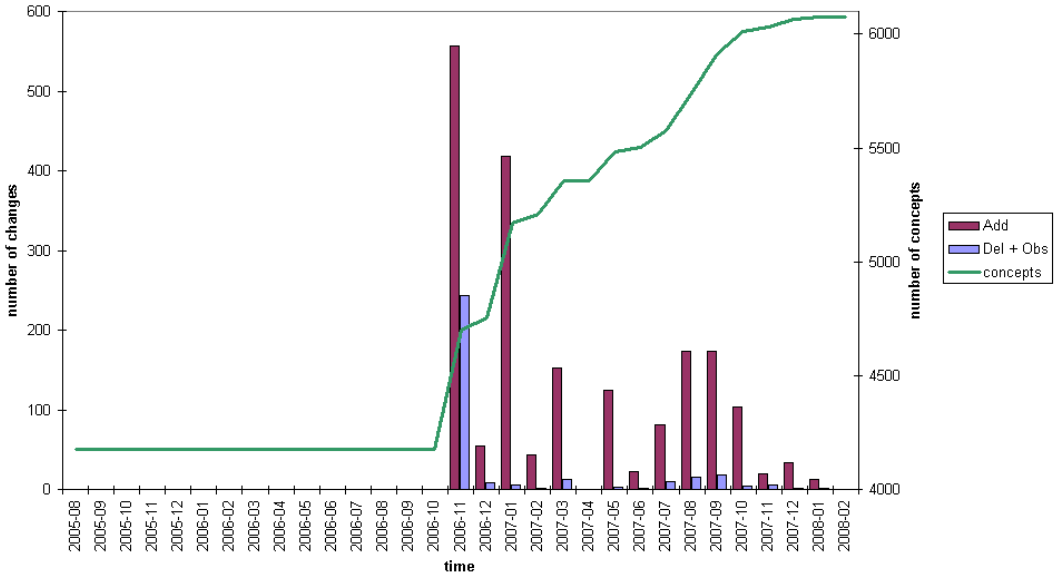


Abbildung A.13: Evolution von Mammalian Phenotype Ontology

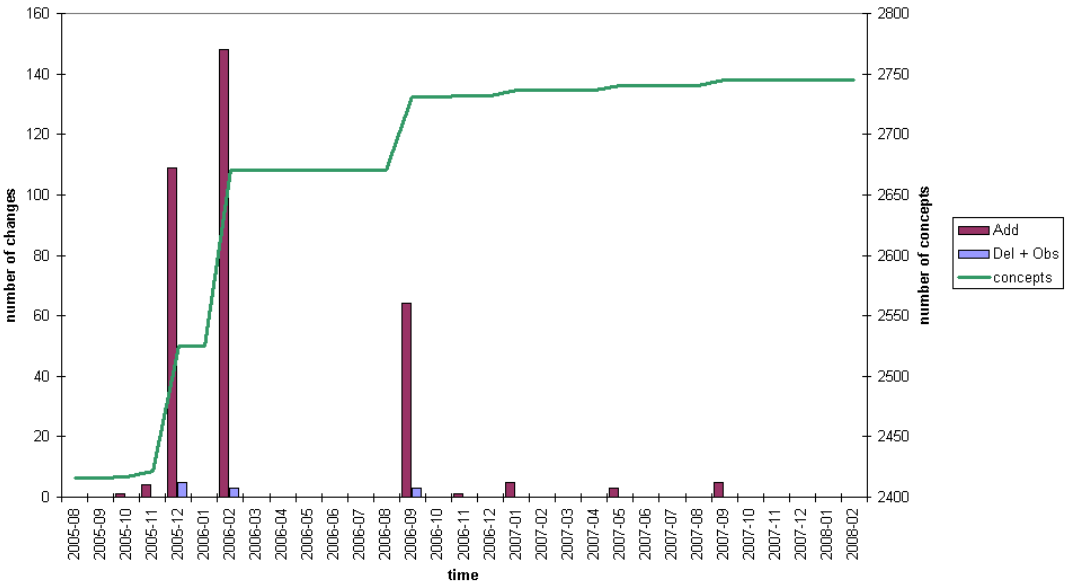


Abbildung A.14: Evolution von Adult Mouse Anatomy

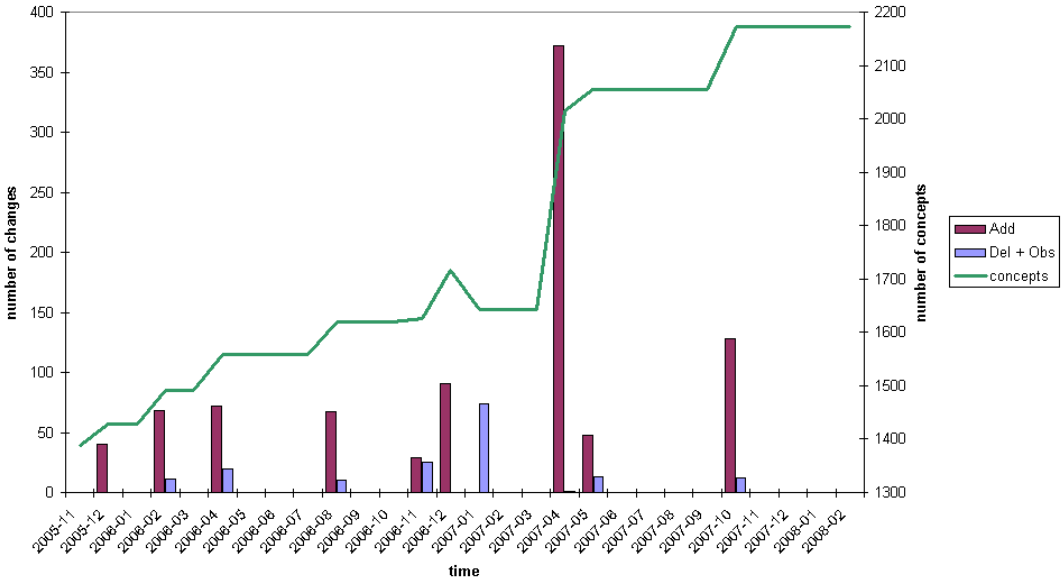


Abbildung A.15: Evolution von Zebrafish Anatomy

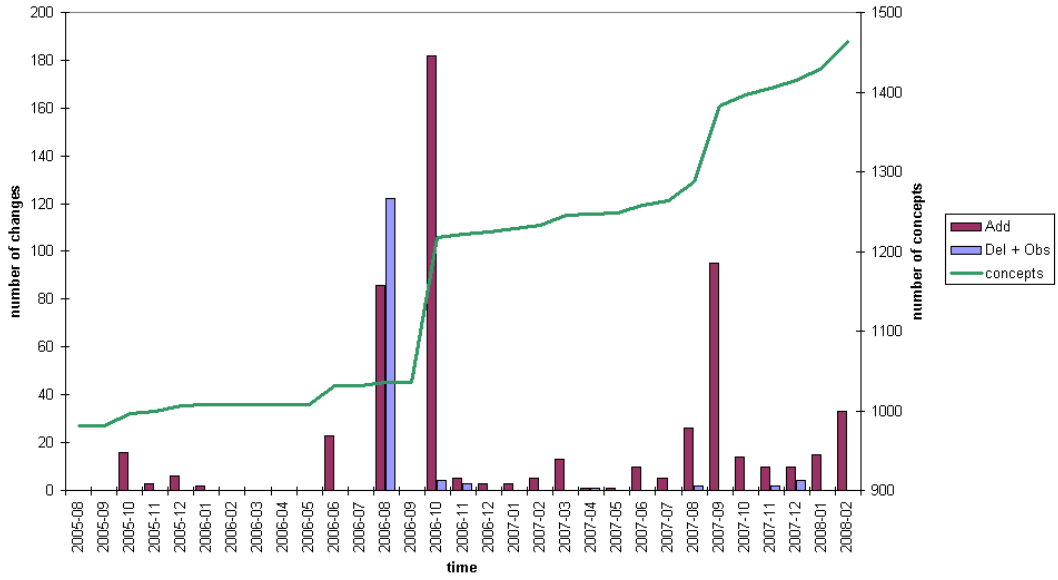


Abbildung A.16: Evolution von Sequence Ontology

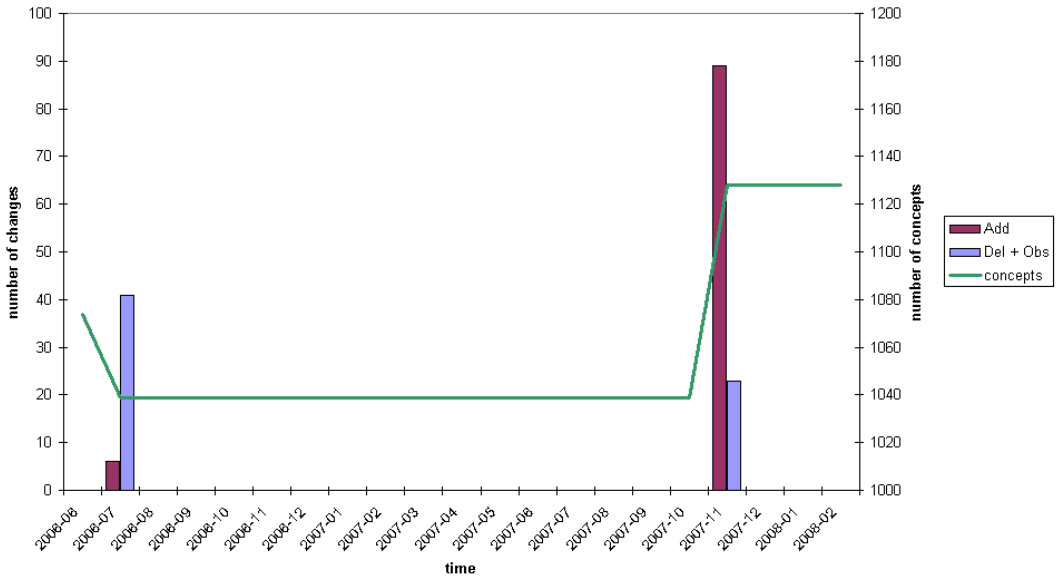


Abbildung A.17: Evolution von Protein Modification Ontology

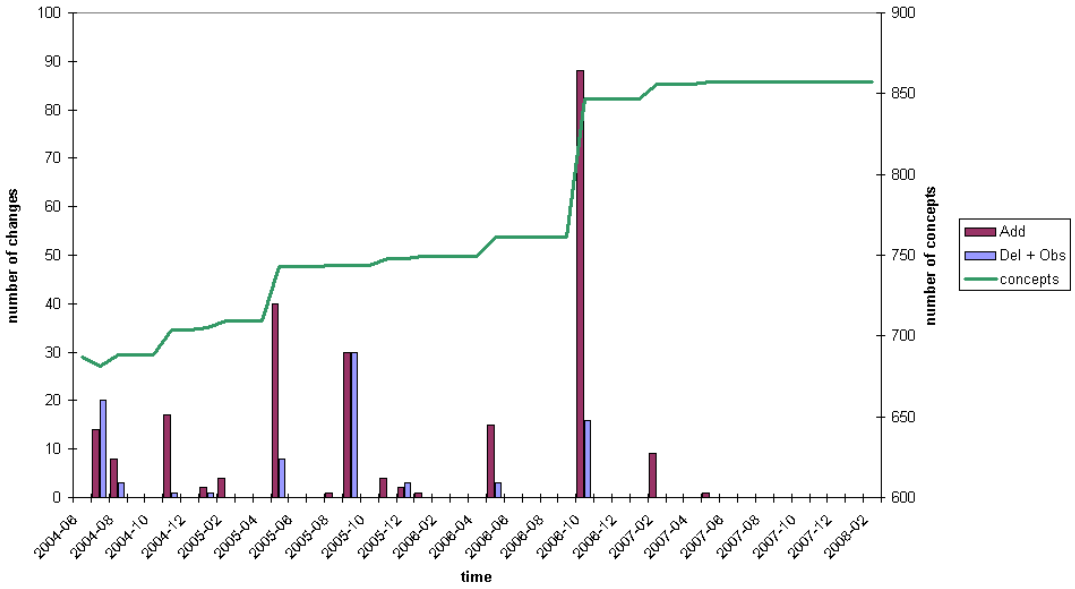


Abbildung A.18: Evolution von CellType Ontology

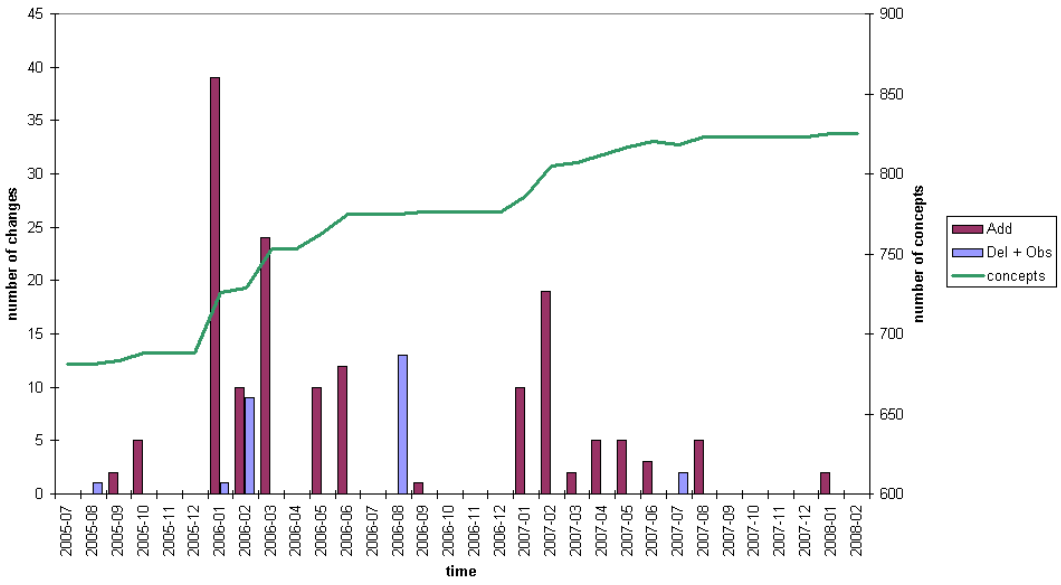


Abbildung A.19: Evolution von PlantStructure Ontology

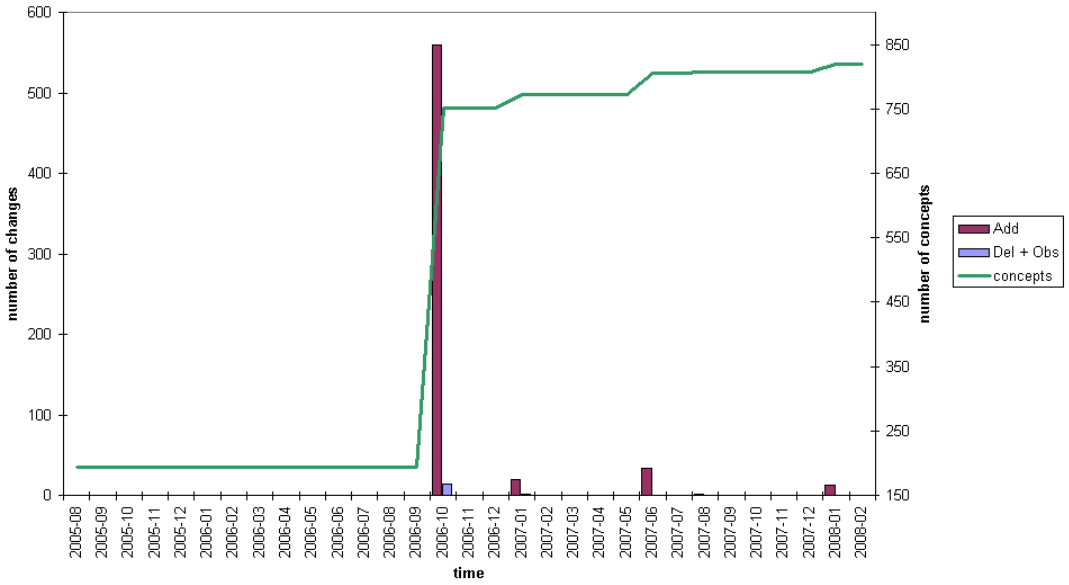


Abbildung A.20: Evolution von ProteinProtein Interaction Ontology

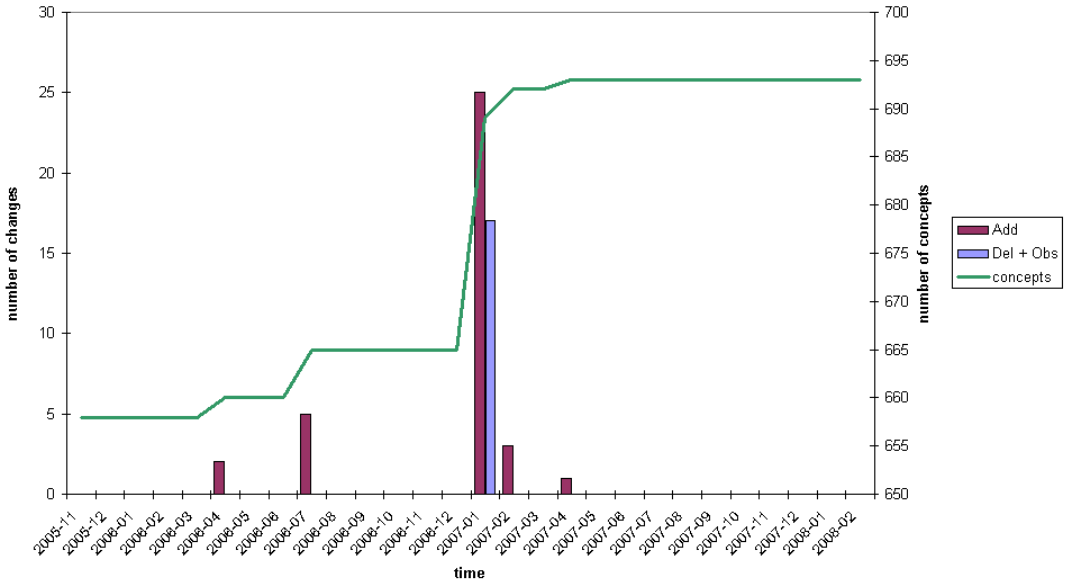


Abbildung A.21: Evolution von FlyBase Controlled Vocabulary

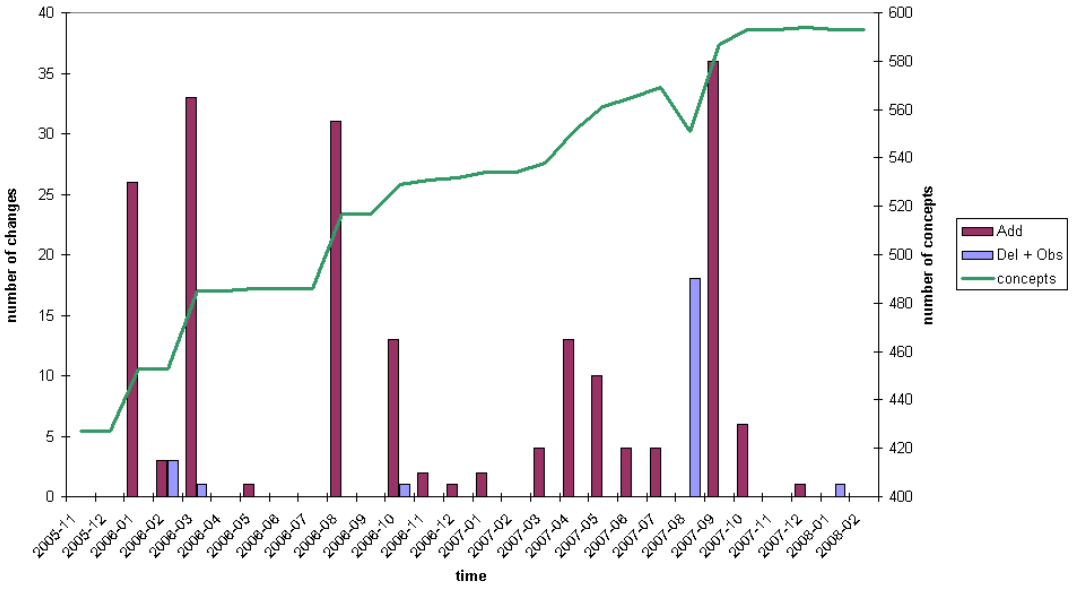


Abbildung A.22: Evolution von Pathway Ontology

B

Regelbasierte Änderungsbestimmung

B.1 Änderungsoperationen und Inverse

Die nachfolgende Tabelle fasst allen verwendeten Änderungsoperationen sowie deren Inverse für den regelbasierten Diff Algorithmus aus Kapitel 5 zusammen.

Änderungsoperation	*Inverse Änderungsoperation*
$addC(c)$	$delC(c)$
$delC(c)$	$addC(c)$
$mapC(c1, c2)$	$mapC(c2, c1)$
$addR(r)$	$delR(r)$
$delR(r)$	$addR(r)$
$mapR(r1, r2)$	$mapR(r2, r1)$
$addA(a)$	$delA(a)$
$delA(a)$	$addA(a)$
$mapA(a1, a2)$	$mapA(a2, a1)$
$substitute(c1, c2)$	$substitute(c2, c1)$
$move(c, c\_to, c\_from)$	$move(c, c\_from, c\_to)$
$toObsolete(c)$	$revokeObsolete(c)$
$revokeObsolete(c)$	$toObsolete(c)$
$addLeaf(c, C\_Parents)$	$delLeaf(c, C\_Parents)$
$delLeaf(c, C\_Parents)$	$addLeaf(c, C\_Parents)$
$merge(Source\_C, target\_c)$	$split(target\_c, Source\_C)$
$split(source\_c, Target\_C)$	$merge(Target\_C, source\_c)$
$addSubGraph(c\_root, C\_Sub)$	$delSubGraph(c\_root, C\_Sub)$
$delSubGraph(c\_root, C\_Sub)$	$addSubGraph(c\_root, C\_Sub)$

Tabelle B.1: Änderungsoperationen und zugehörige Inverse

B.2 COG Regeln

Die nachfolgenden Tabellen listen alle für den vorgestellten Diff Algorithmus verwendeten Regeln gruppiert nach ihrem Typ (b-COG, c-COG, a-COG). Für Konzepte werden die Variablen a, b, c, d und e verwendet. Konzeptmengen werden mit den Großbuchstaben A und B gekennzeichnet. Die Variablen r und s bezeichnen Beziehungen zwischen Konzepten, wohingegen p und q Attribute repräsentieren.

ID	*Regel*
b_1	$c \in O_{new} \wedge \nexists a(a \in O_{old} \wedge matchC(a,c))$ $\rightarrow$ **create**$[addC(c)]$
b_2	$c \in O_{old} \wedge \nexists a(a \in O_{new} \wedge matchC(c,a))$ $\rightarrow$ **create**$[delC(c)]$
b_3	$a \in O_{old} \wedge b \in O_{new} \wedge matchC(a,b) \wedge a \neq b$ $\rightarrow$ **create**$[mapC(a,b)]$
b_4	$a \in O_{old}, O_{new} \wedge matchC(a,a) \wedge \exists b(b \in O_{new} \wedge matchC(a,b) \wedge a \neq b)$ $\rightarrow$ **create**$[mapC(a,a)]$
b_5	$a \in O_{old}, O_{new} \wedge matchC(a,a) \wedge \exists b(b \in O_{old} \wedge matchC(b,a) \wedge a \neq b)$ $\rightarrow$ **create**$[mapC(a,a)]$
b_6	$r \in O_{new} \wedge r \notin O_{old}$ $\rightarrow$ **create**$[addR(r)]$
b_7	$r \in O_{old} \wedge r \notin O_{new}$ $\rightarrow$ **create**$[delR(r)]$
b_8	$r \in O_{old} \wedge s \in O_{new} \wedge delR(r) \wedge addR(s) \wedge r_{source} = s_{source} \wedge$ $r_{target} = s_{target} \wedge r_{type} \neq s_{type}$ $\rightarrow$ **create**$[mapR(r,s)]$, **eliminate**$[delR(r), addR(s)]$
b_9	$p \in O_{new} \wedge p \notin O_{old}$ $\rightarrow$ **create**$[addA(p)]$
b_{10}	$p \in O_{old} \wedge p \notin O_{new}$ $\rightarrow$ **create**$[delA(p)]$
b_{11}	$p \in O_{old} \wedge q \in O_{new} \wedge delA(p) \wedge addA(q) \wedge p_{concept} = q_{concept} \wedge$ $p_{name} = q_{name} \wedge p_{value} \neq q_{value}$ $\rightarrow$ **create**$[mapA(p,q)]$, **eliminate**$[delA(p), addA(q)]$

Tabelle B.2: Liste aller b-COG Regeln

ID	*Regel*
c_1	$a \in O_{old} \wedge b \in O_{new} \wedge mapC(a,b) \wedge a \neq b \wedge$ $\nexists c(c \in O_{new} \wedge mapC(a,c) \wedge b \neq c) \wedge \nexists d(d \in O_{old} \wedge mapC(d,b) \wedge a \neq d)$ $\rightarrow$ **create**$[substitute(a,b)]$, **eliminate**$[mapC(a,b)]$
c_2	$r \in O_{old} \wedge s \in O_{new} \wedge delR(r) \wedge addR(s) \wedge r_{source} = s_{source} \wedge$ $r_{target} \neq s_{target} \wedge r_{type} = s_{type}$ $\rightarrow$ **create**$[move(r_{source}, r_{target}, s_{target})]$, **eliminate**$[delR(r), addR(s)]$
c_3	$p \in O_{old} \wedge q \in O_{new} \wedge mapA(p,q) \wedge p_{concept} = q_{concept} \wedge p_{name} =' obsolete' \wedge$ $p_{name} = q_{name} \wedge p_{value} =' false' \wedge q_{value} =' true'$ $\rightarrow$ **create**$[toObsolete(p_{concept})]$, **eliminate**$[mapA(p,q)]$
c_4	$p \in O_{old} \wedge q \in O_{new} \wedge mapA(p,q) \wedge p_{concept} = q_{concept} \wedge p_{name} =' obsolete' \wedge$ $p_{name} = q_{name} \wedge p_{value} =' true' \wedge q_{value} =' false'$ $\rightarrow$ **create**$[revokeObsolete(p_{concept})]$, **eliminate**$[mapA(p,q)]$
c_5	$a, r \in O_{new} \wedge addC(a) \wedge addR(r) \wedge a = r_{source} \wedge$ $\nexists s(s \in O_{new} \wedge addR(s) \wedge a = s_{target})$ $\rightarrow$ **create**$[addLeaf(a, \{r_{target}\})]$, **eliminate**$[addC(a), addR(r)]$
c_6	$a, r \in O_{old} \wedge delC(a) \wedge delR(r) \wedge a = r_{source} \wedge$ $\nexists s(s \in O_{old} \wedge delR(s) \wedge a = s_{target})$ $\rightarrow$ **create**$[delLeaf(a, \{r_{target}\})]$, **eliminate**$[delC(a), delR(r)]$
c_7	$a, b \in O_{old} \wedge c \in O_{new} \wedge mapC(a,c) \wedge mapC(b,c) \wedge a \neq b \wedge$ $\nexists d(d \in O_{new} \wedge mapC(a,d) \wedge c \neq d) \wedge \nexists e(e \in O_{new} \wedge mapC(b,e) \wedge c \neq e)$ $\rightarrow$ **create**$[merge(\{a\}, c), merge(\{b\}, c)]$, **eliminate**$[mapC(a,c), mapC(b,c)]$
c_8	$c \in O_{old} \wedge a, b \in O_{new} \wedge mapC(c,a) \wedge mapC(c,b) \wedge a \neq b \wedge$ $\nexists d(d \in O_{old} \wedge mapC(d,a) \wedge c \neq d) \wedge \nexists e(e \in O_{old} \wedge mapC(e,b) \wedge c \neq e)$ $\rightarrow$ **create**$[split(c, \{a\}), split(c, \{b\})]$, **eliminate**$[mapC(c,a), mapC(c,b)]$
c_9	$a, b \in O_{new} \wedge B \subseteq O_{new} \wedge addC(a) \wedge addLeaf(b,B) \wedge a \in B$ $\rightarrow$ **create**$[addSubGraph(a, \{b\})]$, **eliminate**$[addC(a), addLeaf(b,B)]$
c_{10}	$a, b \in O_{old} \wedge B \subseteq O_{old} \wedge delC(a) \wedge delLeaf(b,B) \wedge a \in B$ $\rightarrow$ **create**$[delSubGraph(a, \{b\})]$, **eliminate**$[delC(a), delLeaf(b,B)]$

Tabelle B.3: Liste aller c-COG Regeln

ID	*Regel*
a_1	$a \in O_{new} \wedge A, B \subseteq O_{new} \wedge addLeaf(a, A) \wedge addLeaf(a, B) \wedge A \neq B$ $\rightarrow$ **create**$[addLeaf(a, A \cup B)]$, **eliminate**$[addLeaf(a, A), addLeaf(a, B)]$
a_2	$a \in O_{old} \wedge A, B \subseteq O_{old} \wedge delLeaf(a, A) \wedge delLeaf(a, B) \wedge A \neq B$ $\rightarrow$ **create**$[delLeaf(a, A \cup B)]$, **eliminate**$[delLeaf(a, A), delLeaf(a, B)]$
a_3	$c \in O_{new} \wedge A, B \subseteq O_{old} \wedge merge(A, c) \wedge merge(B, c) \wedge A \neq B$ $\rightarrow$ **create**$[merge(A \cup B, c)]$, **eliminate**$[merge(A, c), merge(B, c)]$
a_4	$c \in O_{old} \wedge A, B \subseteq O_{new} \wedge split(c, A) \wedge split(c, B) \wedge A \neq B$ $\rightarrow$ **create**$[split(c, A \cup B)]$, **eliminate**$[split(c, A), split(c, B)]$
a_5	$a, b, r \in O_{new} \wedge A \subseteq O_{new} \wedge addSubGraph(a, A) \wedge addC(b) \wedge addR(r) \wedge$ $r_{source} = a \wedge r_{target} = b$ $\rightarrow$ **create**$[addSubGraph(b, \{a\} \cup A)]$, **eliminate**$[addSubGraph(a, A), addC(b), addR(r)]$
a_6	$a \in O_{new} \wedge A, B \subseteq O_{new} \wedge addSubGraph(a, A) \wedge addSubGraph(a, B) \wedge$ $A \neq B$ $\rightarrow$ **create**$[addSubGraph(a, A \cup B)]$, **eliminate**$[addSubGraph(a, A), addSubGraph(a, B)]$
a_7	$a, b, r \in O_{new} \wedge A, B \subseteq O_{new} \wedge addSubGraph(a, A) \wedge$ $addSubGraph(b, B) \wedge addR(r) \wedge r_{source} = a \wedge (r_{target} = b \vee r_{target} \in B)$ $\rightarrow$ **create**$[addSubGraph(b, \{a\} \cup A \cup B)]$, **eliminate**$[addSubGraph(a, A), addSubGraph(b, B), addR(r)]$
a_8	$a, b, r \in O_{old} \wedge A \subseteq O_{old} \wedge delSubGraph(a, A) \wedge delC(b) \wedge delR(r) \wedge$ $r_{source} = a \wedge r_{target} = b$ $\rightarrow$ **create**$[delSubGraph(b, \{a\} \cup A)]$, **eliminate**$[delSubGraph(a, A), delC(b), delR(r)]$
a_9	$a \in O_{old} \wedge A, B \subseteq O_{old} \wedge delSubGraph(a, A) \wedge delSubGraph(a, B) \wedge$ $A \neq B$ $\rightarrow$ **create**$[delSubGraph(a, A \cup B)]$, **eliminate**$[delSubGraph(a, A), delSubGraph(a, B)]$
a_{10}	$a, b, r \in O_{old} \wedge A, B \subseteq O_{old} \wedge delSubGraph(a, A) \wedge$ $delSubGraph(b, B) \wedge delR(r) \wedge r_{source} = a \wedge (r_{target} = b \vee r_{target} \in B)$ $\rightarrow$ **create**$[delSubGraph(b, \{a\} \cup A \cup B)]$, **eliminate**$[delSubGraph(a, A), delSubGraph(b, B), delR(r)]$

Tabelle B.4: Liste aller a-COG Regeln

B.3 Diff Algorithmus für Beispiel

Dieser Abschnitt zeigt einen Durchlauf des in Kapitel 5 vorgestellten Diff Algorithmus auf dem in Abb. 5.1 dargestellten motivierenden Beispiel. Das Match-Mapping umfasst die folgenden Korrespondenzen:

- $matchC$(Drives&Storage, Drives&Storage)
- $matchC$(Hard Disc Drives, Hard Disc Drives)
- $matchC$(Optical Disc Drives, Optical Disc Drives)
- $matchC$(DVD+/-RW, DVD+/-RW)
- $matchC$(Other, Other)
- $matchC$($3^1/_2$, $3^1/_2$)
- $matchC$($2^1/_2$, $2^1/_2$)
- $matchC$(1.8, 1.8)
- $matchC$(DVD-ROM, Other)
- $matchC$(CD-RW, Other)

Die nachfolgenden Tabellen zeigen wie durch Anwendung der COG Regeln (siehe B.2) die Diff Evolution-Mappings $diff_{basic}$ und $diff_{compact}$ bestimmt werden. Änderungen wurden durch die in Spalte eins gezeigten Regeln erzeugt. Grau hinterlegte Änderungen sind nur temporär vorhanden, d. h. sie wurden zwar erzeugt, es stellte sich jedoch heraus, dass sie von einer komplexeren Änderung abgedeckt werden. Die entsprechende Regel zur Löschung wird in Spalte drei angegeben. Alle Änderungen in Tab. B.5 gehören zu $diff_{basic}$. Alle weiß hinterlegten Änderungen in den Tab. B.5, B.6 und B.7 sind Bestandteil des finalen $diff_{compact}$ Evolution-Mapping.

Erzeugt durch Regel	*Erzeugte Änderungsoperation*	*Gelöscht durch Regel*
b_1	$addC$(HD-DVD)	c_5
b_1	$addC$(Blu-ray)	c_5
b_1	$addC$(Notebook)	
b_1	$addC$(Solid State Disks)	a_5
b_1	$addC$(SLC)	c_9
b_1	$addC$(MLC)	c_9
b_1	$addC$(1.3)	c_5
b_1	$addC$(0.85)	c_5
b_3	$mapC$(DVD-ROM,Other)	c_7
b_3	$mapC$(CD-RW,Other)	c_7
b_3	$mapC$(Other,Other)	c_7
b_6	$addR$(HD-DVD,subCatOf,Optical Disc Drives)	c_5
b_6	$addR$(Blu-ray,subCatOf,Optical Disc Drives)	c_5
b_6	$addR$(Notebook,subCatOf,Hard Disc Drives)	
b_6	$addR$(1.8,subCatOf,Notebook)	c_2
b_6	$addR$(2½,subCatOf,Notebook)	c_2
b_6	$addR$(Solid State Disks,subCatOf,Drives & Storage)	
b_6	$addR$(SLC,subCatOf,Solid State Disks)	a_5
b_6	$addR$(MLC,subCatOf,Solid State Disks)	a_5
b_6	$addR$(1.3,subCatOf,SLC)	c_5
b_6	$addR$(0.85,subCatOf,MLC)	c_5
b_7	$delR$(1.8,subCatOf,Hard Disc Drives)	c_2
b_7	$delR$(2½,subCatOf,Hard Disc Drives)	c_2
b_7	$delR$(DVD-ROM,subCatOf,Optical Disc Drives)	
b_7	$delR$(CD-RW,subCatOf,Optical Disc Drives)	

Tabelle B.5: Anwendung der b-COG Regeln für motivierendes Beispiel

Erzeugt durch Regel	*Erzeugte Änderungsoperation*	*Gelöscht durch Regel*
c_2	*move*(1.8,Hard Disc Drives,Notebook)	
c_2	*move*($2^1/_2$,Hard Disc Drives,Notebook)	
c_5	*addLeaf*(HD-DVD,{Optical Disc Drives})	
c_5	*addLeaf*(Blu-ray,{Optical Disc Drives})	
c_5	*addLeaf*(1.3,{SLC})	c_9
c_5	*addLeaf*(0.85,{MLC})	c_9
c_7	*merge*({DVD-ROM},Other)	a_3
c_7	*merge*({CD-RW},Other)	a_3
c_7	*merge*({Other},Other)	a_3
c_9	*addSubGraph*(SLC,{1.3})	a_5
c_9	*addSubGraph*(MLC,{0.85})	a_5

Tabelle B.6: Anwendung der c-COG Regeln für motivierendes Beispiel

Erzeugt durch Regel	*Erzeugte Änderungsoperation*	*Gelöscht durch Regel*
a_3	*merge*({DVD-ROM,CD-RW},Other)	a_3
a_3	*merge*({DVD-ROM,CD-RW,Other},Other)	
a_5	*addSubGraph*(Solid State Disks,{SLC,1.3})	a_6
a_5	*addSubGraph*(Solid State Disks,{MLC,0.85})	a_6
a_6	*addSubGraph*(Solid State Disks,{SLC,1.3,MLC,0.85})	

Tabelle B.7: Anwendung der a-COG Regeln für motivierendes Beispiel

Literaturverzeichnis

[1] Altova. Altova DiffDog, 2010.

[2] Ambler, S. W., Sadalage, P. J. *Refactoring databases: Evolutionary database design.* Addison-Wesley Professional, 2006.

[3] Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., et al. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nature Genetics 25*, 1 (2000), 25–29.

[4] Ashburner, M., Leser, U., Rebholz-Schuhmann, D., Eds. *Ontologies and Text Mining for Life Sciences: Current Status and Future Perspectives* (2008), vol. 08131 of *Dagstuhl Seminar Proceedings*.

[5] Aumueller, D., Do, H. H., Massmann, S., Rahm, E. Schema and ontology matching with COMA++. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data* (2005), ACM, pp. 906–908.

[6] Bard, J. B. L., Rhee, S. Y. Ontologies in biology: design, applications and future challenges. *Nature Reviews Genetics 5*, 3 (2004), 213–222.

[7] Barrell, D., Dimmer, E., Huntley, R. P., Binns, D., O'Donovan, C., Apweiler, R. The GOA database in 2009–an integrated Gene Ontology Annotation resource. *Nucleic Acids Research 37*, Database Issue (2009), D396–D403.

[8] Barrett, D. J. *MediaWiki.* O'Reilly Media, 2008.

[9] Bellahsene, Z., Bonifati, A., Rahm, E. *Schema Matching and Mapping.* Springer-Verlag, 2011.

[10] Bernstein, P. A. Applying model management to classical meta data problems. In *Proceedings of Conference on Innovative Database Research (CIDR)* (2003), pp. 209–220.

[11] Bernstein, P. A., Halevy, A. Y., Pottinger, R. A. A vision for management of complex models. *ACM SIGMOD Record 29*, 4 (2000), 55–63.

[12] Bernstein, P. A., Melnik, S. Model management 2.0: manipulating richer mappings. In *Proceedings of the 2007 ACM SIGMOD International Conference*

on Management of Data (2007), pp. 1–12.

[13] Bodenreider, O., Stevens, R. Bio-ontologies: current trends and future directions. *Briefings in Bioinformatics 7*, 3 (2006), 256–274.

[14] Boeckmann, B., Bairoch, A., Apweiler, R., Blatter, M. C., Estreicher, A., Gasteiger, E., Martin, M. J., Michoud, K., O'Donovan, C., Phan, I., et al. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Research 31*, 1 (2003), 365–370.

[15] Bonifati, A., Mecca, G., Papotti, P., Velegrakis, Y. Discovery and Correctness of Schema Mapping Transformations. In *Schema Matching and Mapping*, Z. Bellahsene, A. Bonifati, and E. Rahm, Eds. Springer, 2011, ch. 5.

[16] Boyle, E. I., Weng, S., Gollub, J., Jin, H., Botstein, D., Cherry, J. M., Sherlock, G. GO::TermFinder–open source software for accessing Gene Ontology information and finding significantly enriched Gene Ontology terms associated with a list of genes. *Bioinformatics 20*, 18 (2004), 3710–3715.

[17] Buneman, P., Khanna, S., Tajima, K., Tan, W. C. Archiving scientific data. *ACM Transactions on Database Systems 29*, 1 (2004), 2–42.

[18] caBIG Strategic Planning Workspace. The Cancer Biomedical Informatics Grid (caBIG): infrastructure and applications for a worldwide research community. *Studies in health technology and informatics 129*, Pt 1 (2007), 330–334.

[19] Carbon, S., Ireland, A., Mungall, C. J., Shu, S. Q., Marshall, B., Lewis, S., the AmiGO Hub, the Web Presence Working Group. AmiGO: online access to ontology and annotation data. *Bioinformatics 25*, 2 (2009), 288–289.

[20] Cote, R., Jones, P., Apweiler, R., Hermjakob, H. The ontology lookup service, a lightweight cross-platform tool for controlled vocabulary queries. *BMC Bioinformatics 7*, 1 (2006), 97.

[21] Curino, C. A., Moon, H. J., Ham, M. W., Zaniolo, C. The prism workwench: Database schema evolution without tears. In *IEEE 25th International Conference on Data Engineering* (2009), pp. 1523–1526.

[22] Curino, C. A., Moon, H. J., Tanca, L., Zaniolo, C. Schema Evolution in Wikipedia - Toward a Web Information System Benchmark. In *ICEIS (1)* (2008), pp. 323–332.

[23] Curino, C. A., Moon, H. J., Zaniolo, C. Graceful database schema evolution: the prism workbench. *Proceedings of the VLDB Endowment 1*, 1 (2008), 761–772.

[24] Dameron, O., Noy, N. F., Knublauch, H., Musen, M. A. Accessing and manipulating ontologies using web services. In *Proceedings of the ISWC 2004 Workshop on Semantic Web Services* (2004).

[25] Day-Richter, J., Harris, M. A., Haendel, M., Gene Ontology OBO-Edit Working Group, Lewis, S. OBO-Edit–an ontology editor for biologists. *Bioinformatics 23*, 16 (2007), 2198–2200.

[26] Degtyarenko, K., De Matos, P., Ennis, M., Hastings, J., Zbinden, M., McNaught, A., Alcántara, R., Darsow, M., Guedj, M., Ashburner, M. ChEBI: a database and ontology for chemical entities of biological interest. *Nucleic Acids Research 36*, Database Issue (2008), D344–D350.

[27] Domínguez, E., Lloret, J., Rubio, Á. L., Zapata, M. A. Evolving XML schemas and documents using UML class diagrams. In *Database and Expert Systems Applications (DEXA)* (2005), pp. 343–352.

[28] Domínguez, E., Lloret, J., Rubio, Á. L., Zapata, M. A. MeDEA: A database evolution architecture with traceability. *Data & Knowledge Engineering 65*, 3 (2008), 419–441.

[29] Donnelly, K. SNOMED-CT: The advanced terminology and coding system for eHealth. *Studies in Health Technology and Informatics 121* (2006), 279–290.

[30] Euzenat, J., Shvaiko, P. *Ontology matching*. Springer-Verlag New York, 2007.

[31] Fagin, R., Kolaitis, P. G., Popa, L., Tan, W. C. Schema Mapping Evolution through Composition and Inversion. In *Schema Matching and Mapping*, Z. Bellahsene, A. Bonifati, and E. Rahm, Eds. Springer, 2011, ch. 7.

[32] Fallside, D. C., Walmsley, P. XML Schema Part 0: Primer Second Edition. *W3C Recommendation* (2004).

[33] Flicek, P., Aken, B. L., Beal, K., Ballester, B., Caccamo, M., Chen, Y., Clarke, L., Coates, G., Cunningham, F., Cutts, T., et al. Ensembl 2008. *Nucleic Acids Research 36*, Database Issue (2008), D707–D714.

[34] Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., Antoniou, G. Ontology change: classification and survey. *The Knowledge Engineering Review 23*, 2 (2008), 117–152.

[35] Gabel, T., Sure, Y., Völker, J. D3.1.1.a KAON – Ontology Management Infrastructure. Tech. rep., Institute AIFB, University of Karlsruhe, 2004.

[36] Gene Ontology Consortium. The gene ontology project in 2008. *Nucleic Acids Research 36*, Database Issue (2008), D440–D444.

[37] Gentzsch, W., Reinefeld, A. Special Section D-Grid. *Future Generation Computer Systems 25*, 3 (2009), 266–267.

[38] Gross, A., Hartung, M., Kirsten, T., Rahm, E. Estimating the Quality of Ontology-Based Annotations by Considering Evolutionary Changes. In *Data Integration in the Life Sciences (DILS)* (2009), pp. 71–87.

[39] Gruber, T. R. A translation approach to portable ontology specifications. *Knowledge Acquisition 5*, 2 (1993), 199–220.

[40] Grumbling, G., Strelets, V. FlyBase: anatomical data, images and queries. *Nucleic Acids Research 34*, Database Issue (2006), D484–D488.

[41] Guerrini, G., Mesiti, M. X-Evolution: A Comprehensive Approach for XML Schema Evolution. In *DEXA Workshops* (2008), pp. 251–255.

[42] Guerrini, G., Mesiti, M., Rossi, D. Impact of XML schema evolution on valid documents. In *Proceedings of the 7th ACM International Workshop on Web Information and Data Management (WIDM)* (2005), pp. 39–44.

[43] Guerrini, G., Mesiti, M., Sorrenti, M. A. XML Schema Evolution: Incremental Validation and Efficient Document Adaptation. In *XSym* (2007), pp. 92–106.

[44] Haase, P., Sure, Y. D3.1.1.b State-of-the-Art on Ontology Evolution. Tech. rep., Institute AIFB, University of Karlsruhe, 2004.

[45] Hainaut, J. L., Englebert, V., Henrard, J., Hick, J. M., Roland, D. Database evolution: the DB-MAIN approach. *Entity-Relationship Approach—ER'94 Business Modelling and Re-Engineering* (1994), 112–131.

[46] Hartung, M., Gross, A., Kirsten, T., Rahm, E. Discovering Evolving Regions in Life Science Ontologies. In *Data Integration in the Life Sciences (DILS)* (2010), pp. 19–34.

[47] Hartung, M., Gross, A., Rahm, E. Rule-based Generation of Diff Evolution Mappings between Ontology Versions. *CoRR abs/1010.0122* (2010).

[48] Hartung, M., Kirsten, T., Gross, A., Rahm, E. OnEX: Exploring changes in life science ontologies. *BMC Bioinformatics 10*, 1 (2009), 250.

[49] Hartung, M., Kirsten, T., Rahm, E. Analyzing the evolution of life science ontologies and mappings. In *Data Integration in the Life Sciences (DILS)* (2008), pp. 11–27.

[50] Hartung, M., Loebe, F., Herre, H., Rahm, E. A platform for collaborative management of semantic grid metadata. In *Intelligent Distributed Computing, Systems and Applications, Proceedings of the 2nd International Symposium on Intelligent Distributed Computing (IDC)* (2008), pp. 115–125.

[51] Hartung, M., Loebe, F., Herre, H., Rahm, E. Management of evolving semantic grid metadata within a collaborative platform. *Information Sciences 180*, 10 (2010), 1837–1849.

[52] Hartung, M., Rahm, E. A grid middleware for ontology access. In *1st German eScience Conference* (2007).

[53] Hartung, M., Terwilliger, J., Rahm, E. Recent advances in schema and ontology evolution. In *Schema Matching and Mapping*, Z. Bellahsene,

A. Bonifati, and E. Rahm, Eds. Springer, 2011, ch. 6, pp. 149–190.

[54] Hayamizu, T. F., Mangan, M., Corradi, J. P., Kadin, J. A., Ringwald, M. The Adult Mouse Anatomical Dictionary: a tool for annotating and integrating data. *Genome Biology 6*, 3 (2005), R29.

[55] Hick, J. M., Hainaut, J. L. Database application evolution: a transformational approach. *Data & Knowledge Engineering 59*, 3 (2006), 534–558.

[56] Hitzler, P., Krötzsch, M., Rudolph, S., Sure, Y. *Semantic Web: Grundlagen*. Springer, 2007.

[57] IBM. Database version control with IBM Optim Database Administrator V2.2, 2007.

[58] Jakoniene, V., Lambrix, P. Ontology-based integration for bioinformatics. In *VLDB Workshop on Ontologies-based techniques for DataBases and Information Systems (ODBIS)* (2005), pp. 55–58.

[59] Jiang, H., Ho, H., Popa, L., Han, W. S. Mapping-driven XML transformation. In *Proceedings of the 16th International Conference on World Wide Web* (2007), pp. 1063–1072.

[60] Kirsten, T., Hartung, M., Gross, A., Rahm, E. Efficient Management of Biomedical Ontology Versions. In *On the Move to Meaningful Internet Systems: OTM 2009 Workshops* (2009), pp. 574–583.

[61] Kirsten, T., Thor, A., Rahm, E. Instance-based matching of large life science ontologies. In *Proceedings of the 4th International Workshop on Data Integration in the Life Sciences* (2007), pp. 172–187.

[62] Klein, M. *Change management for distributed ontologies*. PhD thesis, Vrije Universiteit Amsterdam, 2004.

[63] Klein, M., Fensel, D., Kiryakov, A., Ognyanov, D. Ontology versioning and change detection on the web. *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web* (2002), 247–259.

[64] Klettke, M. Conceptual XML Schema Evolution - the CoDEX Approach for Design and Redesign. In *BTW Workshops* (2007), pp. 53–63.

[65] Knublauch, H., Fergerson, R. W., Noy, N. F., Musen, M. A. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *The Semantic Web - ISWC 2004: 3rd International Semantic Web Conference* (2004), pp. 229–243.

[66] Kramer, D. Xem: XML evolution management. Master's thesis, Worcester Polytechnic Institute, 2001.

[67] Krefting, D., Bart, J., Beronov, K., Dzhimova, O., Falkner, J., Hartung, M., Hoheisel, A., Knoch, T., Lingner, T., Mohammed, Y., et al. MediGRID: Towards a user friendly secured grid infrastructure.

Future Generation Computer Systems 25, 3 (2009), 326–336.

[68] Lambrix, P., Tan, H., Jakoniene, V., Strömbäck, L. Biological ontologies. In *Semantic Web: Revolutionizing Knowledge Discovery in the Life Sciences*, C. J. O. Baker and K.-H. Cheung, Eds. Springer Verlag, 2007, pp. 85–99.

[69] Lassila, O., McGuinness, D. The role of frame-based representation on the semantic web. *Linköping Electronic Articles in Computer and Information Science 6*, 5 (2001).

[70] Leonardi, E., Hoai, T. T., Bhowmick, S. S., Madria, S. DTD-Diff: A change detection algorithm for DTDs. *Data & Knowledge Engineering 61*, 2 (2007), 384–402.

[71] Leser, U., Naumann, F. *Informationsintegration.* dpunkt.verlag, 2007.

[72] Lipscomb, C. E. Medical subject headings (MeSH). *Bulletin of the Medical Library Association 88*, 3 (2000), 265–266.

[73] Manola, F., Miller, E. RDF Primer. W3C Recommendation, World Wide Web Consortium (W3C), Cambridge, Massachusetts, 2004.

[74] Maule, A., Emmerich, W., Rosenblum, D. S. Impact analysis of database schema changes. In *30th International Conference on Software Engineering (ICSE)* (2008), pp. 451–460.

[75] McCarthy, F. M., Bridges, S. M., Wang, N., Magee, G. B., Williams, W. P., Luthe, D. S., Burgess, S. C. Agbase: a unified resource for functional analysis in agriculture. *Nucleic Acids Research 35*, Database Issue (2007), D599–D603.

[76] McGuinness, D. L., Van Harmelen, F. OWL web ontology language overview. *W3C Recommendation* (2004).

[77] McKusick, V. A. Mendelian Inheritance in Man and its online version, OMIM. *American Journal of Human Genetics 80*, 4 (2007), 588–604.

[78] Melnik, S. *Generic model management: concepts and algorithms.* Springer Verlag New York, 2004.

[79] Mesiti, M., Celle, R., Sorrenti, M., Guerrini, G. X-evolution: A system for xml schema evolution and document adaptation. In *Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology* (2006), pp. 1143–1146.

[80] Microsoft. Data-tier Applications in SQL Server 2008 R2, 2010.

[81] Microsoft. Using SQL Server Management Studio, 2010.

[82] Moreira, D. A., Musen, M. A. OBO to OWL: a protege OWL tab to read/save OBO ontologies. *Bioinformatics 23*, 14 (2007), 1868–1870.

[83] Moro, M. M., Malaika, S., Lim, L. Preserving XML queries during schema evolution. In *Proceedings of the 16th International Conference on World Wide Web* (2007), pp. 1341–1342.

[84] Müller, H., Buneman, P., Koltsidas, I. XArch: archiving scientific and reference data. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (2008), pp. 1295–1298.

[85] Nadkarni, P. M., Marenco, L., Chen, R., Skoufos, E., Shepherd, G., Miller, P. Organization of heterogeneous scientific data using the EAV/CR representation. *Journal of the American Medical Informatics Association 6*, 6 (1999), 478–493.

[86] Noy, N. F. Ontology Management with the Prompt Plugin. In *7th International Protégé Conference. Stanford Center for Biomedical Informatics Research, CA* (2004).

[87] Noy, N. F., Chugh, A., Liu, W., Musen, M. A. A framework for ontology evolution in collaborative environments. In *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference* (2006), pp. 544–558.

[88] Noy, N. F., Klein, M. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems 6*, 4 (2004), 428–440.

[89] Noy, N. F., Kunnatur, S., Klein, M., Musen, M. A. Tracking changes during ontology evolution. In *The Semantic Web - ISWC 2004: Third International Semantic Web Conference* (2004), pp. 259–273.

[90] Noy, N. F., Musen, M. A. Promptdiff: A fixed-point algorithm for comparing ontology versions. In *Proceedings of the National Conference on Artificial Intelligence* (2002), pp. 744–750.

[91] Noy, N. F., Shah, N. H., Whetzel, P. L., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Rubin, D. L., Storey, M. A., Chute, C. G., et al. BioPortal: ontologies and integrated data resources at the click of a mouse. *Nucleic Acids Research 37*, Web Server Issue (2009), W170–W173.

[92] Oracle. Oracle Database 10g Release 2 Online Data Reorganization & Redefinition, 2005.

[93] Oracle. Edition-Based Redefinition, 2009.

[94] Oracle. Oracle Enterprise Manager Getting Started with Oracle Change Management Pack, 2010.

[95] Osborne, J., Flatow, J., Holko, M., Lin, S., Kibbe, W., Zhu, L., Danila, M., Feng, G., Chisholm, R. Annotating the human genome with Disease Ontology. *BMC Genomics 10*, Suppl 1 (2009), S6.

[96] Papastefanatos, G., Vassiliadis, P., Simitsis, A., Aggistalis, K., Pechlivani, F., Vassiliou, Y. Language Extensions for the Automation

of Database Schema Evolution. In *10th International Conference on Enterprise Information Systems (ICEIS)* (2008), pp. 74–81.

[97] Papastefanatos, G., Vassiliadis, P., Simitsis, A., Vassiliou, Y. HECATAEUS: Regulating schema evolution. In *26th International Conference on Data Engineering (ICDE)* (2010), pp. 1181–1184.

[98] Papavassiliou, V., Flouris, G., Fundulaki, I., Kotzinos, D., Christophides, V. On detecting high-level changes in RDF/S KBs. In *The Semantic Web - ISWC 2009, 8th International Semantic Web Conference* (2009), pp. 473–488.

[99] Park, Y. R., Park, C. H., Kim, J. H. Gochase: correcting errors from gene ontology-based annotations for gene products. *Bioinformatics 21*, 6 (2005), 829–831.

[100] Plessers, P., Troyer, O. Ontology change detection using a version log. In *The Semantic Web - ISWC 2005, 4th International Semantic Web Conference* (2005), pp. 578–592.

[101] Plessers, P., Troyer, O. D., Casteleyn, S. Understanding ontology evolution: A change detection approach. *Web Semantics: Science, Services and Agents on the World Wide Web 5*, 1 (2007), 39–49.

[102] Prüfer, K., Muetzel, B., Do, H. H., Weiss, G., Khaitovich, P., Rahm, E., Pääbo, S., Lachmann, M., Enard, W. FUNC: a package for detecting significant associations between gene sets and ontological annotations. *BMC Bioinformatics 8*, 1 (2007), 41.

[103] Pruitt, K. D., Tatusova, T., Maglott, D. R. NCBI Reference Sequence (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Research 33*, Database Issue (2005), D501–D504.

[104] Rahm, E., Bernstein, P. A. A survey of approaches to automatic schema matching. *The VLDB Journal 10*, 4 (2001), 334–350.

[105] Rahm, E., Bernstein, P. A. An online bibliography on schema evolution. *ACM SIGMOD Record 35*, 4 (2006), 30–31.

[106] Rosse, C., Mejino, J. L. V. A reference ontology for biomedical informatics: the Foundational Model of Anatomy. *Journal of Biomedical Informatics 36*, 6 (2003), 478–500.

[107] Shvaiko, P., Euzenat, J. A survey of schema-based matching approaches. *Journal on Data Semantics IV* (2005), 146–171.

[108] Shvaiko, P., Euzenat, J., Giunchiglia, F., Stuckenschmidt, H., Noy, N. F., Rosenthal, A., Eds. *Proceedings of the 4th International Workshop on Ontology Matching (OM-2009) collocated with the 8th International Semantic Web Conference (ISWC-2009)* (2009), vol. 551 of *CEUR Workshop Proceedings*, CEUR-WS.org.

[109] Sioutos, N., Coronado, S., Haber, M. W., Hartel, F. W., Shaiu, W. L., Wright, L. NCI Thesaurus: A semantic model integrating cancer-related clinical and molecular information. *Journal of Biomedical Informatics 40*, 1 (2007), 30–43.

[110] Smith, B., Ashburner, M., Rosse, C., Bard, J., Bug, W., Ceusters, W., Goldberg, L. J., Eilbeck, K., Ireland, A., Mungall, C. J., et al. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology 25*, 11 (2007), 1251–1255.

[111] Sprague, J., Bayraktaroglu, L., Clements, D., Conlin, T., Fashena, D., Frazer, K., Haendel, M., Howe, D. G., Mani, P., Ramachandran, S., et al. The Zebrafish Information Network: the zebrafish model organism database. *Nucleic Acids Research 34*, Database Issue (2006), D581–D585.

[112] Stevens, R., Baker, P., Bechhofer, S., Ng, G., Jacoby, A., Paton, N. W., Goble, C. A., Brass, A. TAMBIS: transparent access to multiple bioinformatics information sources. *Bioinformatics 16*, 2 (2000), 184–186.

[113] Stojanovic, L. *Methods and tools for ontology evolution.* PhD thesis, University of Karlsruhe, 2004.

[114] Stojanovic, L., Maedche, A., Motik, B., Stojanovic, N. User-driven ontology evolution management. *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web* (2002), 133–140.

[115] Su, H., Kramer, D., Chen, L., Claypool, K. T., Rundensteiner, E. A. XEM: Managing the evolution of XML Documents. In *11th International Workshop on Research Issues in Data Engineering (RIDE)* (2001), pp. 103–110.

[116] Thor, A., Hartung, M., Gross, A., Kirsten, T., Rahm, E. An evolution-based approach for assessing ontology mappings-A case study in the life sciences. In *Proc. Conference of the Business, Technology and Web (BTW)* (2009), pp. 277–286.

[117] Türker, C. Schema Evolution in SQL-99 and Commercial (Object-) Relational DBMS. *Database Schema Evolution and Meta-Modeling* (2006), 1–32.

[118] Velegrakis, Y., Miller, R. J., Popa, L. Mapping adaptation under evolving schemas. In *Proceedings of the 29th International Conference on Very Large Data Bases* (2003), pp. 584–595.

[119] Volz, R., Oberle, D., Staab, S., Motik, B. KAON server-a semantic web management system. In *Alternate Track Proceedings of the 12th International World Wide Web Conference* (2003), pp. 20–24.

[120] W3C. XML Schema Versioning Use Cases. Framework for discussion of versioning, 2006.

[121] Wang, J. Z., Ali, F., Appaneravanda, R. A Web Service for Efficient Ontology Comparison. In *Proceedings of the IEEE International Conference on Web Services (ICWS)* (2005), pp. 843–844.

[122] Yang, Z., Zhang, D., Ye, C. Ontology analysis on complexity and evolution based on conceptual model. In *Data Integration in the Life Sciences (DILS)* (2006), pp. 216–223.

[123] Yildiz, B. Ontology evolution and versioning. Tech. rep., E188 - Institut für Softwaretechnik und Interaktive Systeme; Technische Universität Wien, 2006.

[124] Yu, C., Popa, L. Semantic adaptation of schema mappings when schemas evolve. In *Proceedings of the 31st International Conference on Very Large Data Bases* (2005), pp. 1006–1017.

[125] Zeeberg, B. R., Feng, W., Wang, G., Wang, M. D., Fojo, A. T., Sunshine, M., Narasimhan, S., Kane, D. W., Reinhold, W. C., Lababidi, S., et al. GoMiner: a resource for biological interpretation of genomic and proteomic data. *Genome Biology 4*, 4 (2003), R28.

[126] Zhang, B., Schmoyer, D., Kirov, S., Snoddy, J. GOTree Machine(GOTM): a web-based platform for interpreting sets of interesting genes using Gene Ontology hierarchies. *BMC Bioinformatics 5*, 1 (2004), 16.