Semiautomatischer Aufbau und Anwendung von EFGT-Netzen

– Beiträge zum Lebenszyklus von EFGT-Netzen –

Eduardo Torres Schumann

München 2009

Semiautomatischer Aufbau und Anwendung von EFGT-Netzen

– Beiträge zum Lebenszyklus von EFGT-Netzen –

Inaugural-Dissertation
zur Erlangung des Doktorgrades
der Philosophie der Ludwig–Maximilians–Universität
München
am Centrum für Informations- und Sprachverarbeitung (CIS)

vorgelegt von
Eduardo Torres Schumann
aus Sevilla (Spanien)

München, den 26. Februar 2009

Erstgutachter: Prof. Klaus U. Schulz
Zweitgutachter: Prof. Franz Guenthner
Tag der mündlichen Prüfung: 20. Juli 2009

Danksagung

An dieser Stelle möchte ich einigen der Personen ausdrücklich danken, die mich in der Zeit meiner Dissertation begleitet und auf vielfältige Weise unterstützt haben.

Prof. Kai-Uwe Kühnberger und Prof. Uwe Mönnich nahmen mich im DFG-Projekt *Adaptive Ontologies on Extreme Markup Languages* freundlich auf und ermöglichten mir gleich am Anfang einen fachlichen Blick über den Tellerrand hinaus und mein Auskommen im ersten Jahr. Besonders herzlich möchte ich mich bei Uwe Mönnich für seine in jeder Hinsicht unterstützende Art bedanken.

Vielen Dank an Annette Gotscharek und Uli Reffle für ihr teilnehmendes Interesse, welches mich immer wieder auf neue Gedanken gebracht hat. Ohne die von Levin Brunner bereitgestellte Infrastruktur hätte ich viele der Programme nicht implementieren können. Vielen Dank für die stete Unterstützung und die intensive Zeit. An vielen Stellen habe ich auch an die Arbeit von Felix Weigel anknüpfen können, mit dem ich in der anfänglichen Zeit zusammengearbeitet habe. Christoph Ringlstetter hat das Projekt mit der University of Lethbridge organisiert, in dem ein Teil der vorliegenden Arbeit eine praktische Anwendung fand.

Ganz besonders verpflichet fühle ich mich meinem Doktorvater Klaus U. Schulz, der neben seiner inspirierenden Betreuung immer ein offenes Ohr nicht nur für fachliche Angelegenheiten hatte. Er hat sich immer darum gekümmert, dass ich meine Arbeit zu den günstigen Bedingungen fortsetzen konnte, die er mitgeschaffen hat. Hierfür möchte ich mich herzlich bei ihm bedanken.

Prof. Franz Guenthner möchte ich dafür danken, dass er mich am Anfang meines Studiums der Computerlinguistik mit seiner Begeisterung für dieses Fach angesteckt hat und für die offene Atmosphäre am CIS, die er maßgeblich geprägt hat.

Einen ganz besonderen Dank schulde ich Corinna Wolf für ihre Korrekturarbeiten, aber vor allem für ihre Unterstützung, Geduld und Liebe in dieser Zeit.

Einleitung

Mit der Entwicklung von Ontologien sollen Ressourcen entstehen, die für die unterschiedlichsten Bereiche ein verbindliches Vokabular und wichtige Relationen zwischen dessen Elementen in maschinenlesbarer Form bereitstellen – ein Teil des spezifischen Wissens eines jeden Bereichs. Mit Hilfe dieser Art von Ressourcen soll der Austausch von Wissen zwischen Maschinen untereinander und zwischen Maschine und Mensch wesentlich erleichtert werden sowie sich viele wissensintensive Aufgaben überhaupt automatisieren lassen. Somit werden Ontologien als Schlüsselwerkzeug insbesondere für Anwendungen angesehen, mit denen versucht wird, die Informationsflut im Internet zu beherrschen. Die Schwierigkeiten dabei sind, dass sich das Erfassen von Wissen und die Entwicklung von Ontologien sehr aufwändig gestalten und immer der Realität nachhinken.

EFGT-Netze, eine spezielle Art von Ontologien, die am Centrum für Informations- und Sprachverarbeitung (CIS) der Ludwig-Maximilians-Universität München entwickelt wurde, versuchen einen Teil der natürlichen Sprache zu erfassen. Dabei geht es um sprachliche Ausdrücke, mit denen Gegenstände, Organisationen, Ereignisse, Personen – oft als Entitäten der Wirklichkeit bezeichnet – benannt werden. Mit Ausdrücken dieser Art wie etwa *Mount Everest*, *Azteken*, *Alpinsport*, *Legoland* oder *Gerhard Polt* verbinden Menschen eine schier unendliche Reihe von Informationen und Themen – das gesamte assoziierte Hintergrundwissen, das wesentlich für das Verständnis von Texten oder generell von sprachlichen Aussagen ist. Ein EFGT-Netz zielt darauf ab, diese Querverbindungen zwischen Ausdrücken und thematischen Zusammenhängen in seiner Struktur zu modellieren. Als Ressource, die dieses Hintergrundwissen zur Verfügung stellt, bieten sich für EFGT-Netze ebenso vielfältige Anwendungen in der automatischen Sprachverarbeitung und der Computerlinguistik.

Das Problem des aufwändigen Aufbauens trifft auf EFGT-Netze als Ontologie ganz besonders zu, da es sich dabei um linguistische, breit angelegte Ontologien handelt. Daraus ergeben sich u.a. folgende Fragestellungen: Wie werden im Allgemeinen Ontologien aufgebaut und welche Lehren lassen sich deshalb für die effiziente Entwicklung von EFGT-Netzen ziehen? Wie lässt sich eine linguistische Ontologie mit unüberschaubar vielen Einträgen überhaupt effizient anlegen? Wie lassen sich extern vorhandene Datenquellen hierfür in ihrer gesamten Größe in ein EFGT-Netz integrieren? Wie lässt sich deren Qualität und Aktualität längerfristig wahren? Die vorliegende Arbeit befasst sich schwerpunktmäßig mit diesen und weiteren damit verbundenen Fragen aus der spezifischen Perspektive, wie sich in der Praxis der Vorgang der Entwicklung von EFGT-Netzen technisch unterstützen und effizienter gestalten lässt.

Die ersten zwei Kapitel bilden die Grundlagen der Arbeit. Kap. 1 gibt einen kurzen Überblick über Ontologien im Allgemeinen und betrachtet verschiedene Strategien, die sich bei deren Entwicklung verfolgen lassen. Auf EFGT-Netze geht Kap. 2 näher ein. Zunächst werden die Ziele und die Motivation sowie der spezifische EFGT-Formalismus vorgestellt, um dann die Vorgehensweise zu beschreiben, mit der in der Praxis EFGT-Netze entwickelt werden. Am Ende dieses zweiten Kapitels werden konkrete technische Ziele definiert.

In den darauffolgenden Kapiteln wird die Umsetzung dieser Ziele diskutiert: Kap. 3 betrachtet das Ziel der Integration von semistrukturierten Daten als grundlegende Maßnahme für den Aufbau eines EFGT-Netzes. Darin wird eine spezielle Sprache entwickelt, mit der sich ausgehend von vorhandenen Daten ein EFGT-Netz erweitern lässt. Es wird ein System vorgestellt, das auf dieser Sprache aufbaut und mit dem Ontologieentwickler in den Prozess der automatischen Integration von Daten eingreifen und ihn überwachen können.

Mit dem thematischen Archivbrowser wird in Kap. 4 eine exemplarische, innovative Anwendung von EFGT-Netzen vorgestellt, in der ein thematisches Inhaltsverzeichnis für das Archiv erzeugt wird. Damit werden Navigation und Suche in der Dokumentensammlung ermöglicht und gleichzeitig ein wichtiges Feedback über den Entwicklungszustand der Ontologie geliefert.

Kap. 5 befasst sich mit den Möglichkeiten, mit Hilfe einer Textsammlung ein EFGT-Netz um relevante Entitäten zu erweitern. Im Zuge dessen wird eine Erweiterung des Archivbrowsers aus Kap. 4 erarbeitet, mit der sich ein EFGT-Netz längerfristig pflegen lässt.

Die Bedeutung dieser Maßnahmen für den Ablauf der Entwicklung von EFGT-Netzen wird in Kap. 6 rückblickend betrachtet. Die Arbeit schließt mit einem Überblick über mögliche Weiterentwicklungen und interessante Fragestellungen für die Zukunft.

> "Wohl ist's ersichtlich", versetzte Don Quijote, "daß du in Sachen der Abenteuer nicht kundig bist; es sind Riesen, und wenn du Furcht hast, mach dich fort von hier und verrichte dein Gebet, während ich zu einem grimmen und ungleichen Kampf mit ihnen schreite."

Don Quijote, Kap. 8 (Cervantes Saavedra, 1985, S. 59)

Inhaltsverzeichnis

Danksagung **iv**

Einleitung **v**

1 Ontologien und *Ontology Engineering* **1**
1.1 Ontologien . . . 1
1.2 Ontologien als Formalismus: Ontologiesprachen . . . 4
1.3 Entwicklung von Ontologien: *Ontology Engineering* . . . 8
1.3.1 Arten von Ontologien . . . 9
1.3.2 Methodologien zur Entwicklung von Ontologien . . . 12
1.3.3 Technologische Unterstützung des Ontologie-Lebenszyklus' . . . 24
1.3.4 Evaluation von Ontologien . . . 28
1.4 Ontologien in der Computerlinguistik und NLP . . . 31
1.5 Schlussfolgerungen und Ausblick . . . 33

2 *Ontology Engineering* für EFGT-Netze **35**
2.1 Überblick über EFGT-Netze . . . 35
2.1.1 Motivation . . . 35
2.1.2 Der EFGT-Formalismus . . . 37
2.1.3 Technik . . . 40
2.1.4 Das CoGE-Netz . . . 41
2.1.5 Anwendungen . . . 42
2.2 *Ontology Engineering* für EFGT-Netze . . . 43
2.2.1 Das CoGE-Netz als Ontologieprojekt . . . 43
2.2.2 Technische Infrastruktur für den Ontologie-Entwickler . . . 45
2.2.3 Der Lebenszyklus eines EFGT-Netzes am Beispiel des CoGE-Netzes 47
2.2.4 Schlussfolgerungen . . . 56
2.3 Ziele . . . 58
2.3.1 Ausblick . . . 59

3 Integration von semi-strukturierten Daten: Das *Upload-Tool* **60**
3.1 Ausgangspunkt: Muster bei der Kodierung von EFGT-Netz-Einträgen . . . 60
3.2 Anforderungen an ein System zur Datenintegration . . . 63

3.3 Eine Sprache zur Integration von Daten in EFGT-Netze 64
3.3.1 Schemata zur Definition generischer EFGT-Netz-Einträge 65
3.3.2 Abgleich und Alinierung generierter Einträge 71
3.3.3 Datenformate und Instanziierung von Schemata 75
3.4 Die Implementierung: Das *Upload-Tool* 83
3.4.1 Allgemeiner Ablauf bei der Verwendung des Upload-Tools 84
3.4.2 Architektur 84
3.4.3 Eine konkrete Syntax für Templates und *Upload-Files* 88
3.4.4 Der Client 90
3.4.5 Anwendungsfälle 99
3.5 Schlussfolgerungen und Ausblick 104

4 Thematische Suche und Navigation in Dokumentarchiven 107
4.1 Motivation für den thematischen Archivbrowser 107
4.2 Online-Navigation in Pressearchiven 109
4.3 Der thematische Archivbrowser 116
4.3.1 Thematische Navigation und Suche 117
4.3.2 Die Architektur des thematischen Archivbrowsers 121
4.3.3 Berechnung eines thematischen Inhaltsverzeichnisses 122
4.4 Bedeutung für den Ontologieentwicklungsprozess 127
4.5 Der thematische Browser als Navigationsmittel für Pressearchive 130
4.6 Schlussfolgerungen und Ausblick 131

5 Dokumentzentrierte Akquisition von Konzepten: Der Browser-Editor 133
5.1 Motivation 133
5.2 Anforderungen an die Funktionalität des Browser-Editors 135
5.2.1 Verschiedene Möglichkeiten für den Ausbau des thematischen Archivbrowsers 135
5.2.2 Fokus auf Akquisition 136
5.2.3 Akquisition der linguistischen Repräsentation 137
5.2.4 Akquisition von Relationen für die Kodierung 138
5.3 Eine Konzeption für die Funktionalität des Browser-Editors 140
5.3.1 Die Benutzeroberfläche 140
5.3.2 Erkennung von Kandidaten während der Indexierung 145
5.3.3 Eine Architektur für den Browser-Editor und Umsetzung der Implementierung 147
5.4 Schlussfolgerungen 153

6 Schlussfolgerungen und Ausblick 155

A Konkrete Syntax der EFGT-Netz-Eintragsschemata 159

Abbildungsverzeichnis

1.1 Ontologiesprachen auf der Semantischen Treppe 6
1.2 Evolution von Ontologiesprachen 7
1.3 Generischer Lebenszyklus einer Ontologie 19

2.1 Syntax der ID-String-Identifikatoren 38
2.2 Weboberfläche zur Graphdarstellung und Navigation in einem EFGT-Netz 46
2.3 Beispiel eines EFGT-Netz-Eintrags im Eintragsformular 47
2.4 Generische und praktische Anwendungsfälle für den Netzentwickler 48
2.5 Phasen im Lebenszyklus eines EFGT-Netzes am Beispiel des CoGE-Netzes 49

3.1 Grammatik zur Spezifikation von Attributlisten 66
3.2 Syntax der erweiterten Identifikatoren $\mathbb{ID}\mathbb{S}\mathrm{tring}^{gen}$ 67
3.3 Eintragsschema zur Kodierung von Bezirken und deren Hauptstädte für schweizerische Kantone (sog. *Bezirkstemplate*) 68
3.4 Mit dem *Bezirkstemplate* generierte Einträge für die erste Zeile in Tab. 3.1 69
3.5 Das *Bezirkstemplate* mit Verweisen auf die Spalten von Tabelle 3.1 77
3.6 Geographische Daten bezüglich der Schweiz als typisierte Felder 78
3.7 Das *Bezirkstemplate* mit Verweisen auf die typisierten Felder in Abb. 3.6 . 78
3.8 Ein Template zur Integration der Taxonomie von Krankheiten in Abb. 3.9 79
3.9 Eine Taxonomie von Krankheiten im Format typisierter Felder 80
3.10 Struktur der in Abb. 3.9 kodierten Krankheitstaxonomie 81
3.11 Zwei XML-Darstellungen der Daten in Tabelle 3.1 82
3.12 Das *Bezirkstemplate* mit XPath-Variablen für den XML-Ausschnitt A in Abb. 3.11 83
3.13 Schematische Darstellung der Architektur des Upload-Tools 86
3.14 Der Client des Upload-Tools 91
3.15 Globale Sicht auf Alinierungsergebnisse 93
3.16 Darstellung einzelner, konfliktfreier Alinierungsergebnisse 94
3.17 Darstellung von Konflikten 94
3.18 Upload-File zur Population der Ontologie mit Personen 100
3.19 Upload-File zur Population der Ontologie mit Komposita 101
3.20 Upload-File zum Ausbau der Ontologie mit einer Taxonomie von Krankheiten 102
3.21 Template zur Erweiterung der linguistischen Repräsentation 103

3.22 Gesamter Ablauf bei der Integration semistrukturierter Daten mit Hilfe des Upload-Tools 105

4.1 Navigationselemente und zusätzliche Dienste auf *sueddeutsche.de* 110
4.2 Thematische Kategorien (*topics*) bei der *New York Times* 112
4.3 Wortwolke als Themen des Tages bei Spiegel ONLINE 113
4.4 Themenbrowser als Zugangsfunktion auf das *Zeit online*-Archiv 114
4.5 Personen- und Organisationenverzeichnis für die tägliche Onlineausgabe von *El País* 115
4.6 Thematisches Inhaltsverzeichnis bei *ElPaís.com* 116
4.7 Die Benutzeroberfläche des thematischen Archivbrowsers 117
4.8 Auto-Vervollständigungsfunktion im Suchfeld des thematischen Archivbrowsers 118
4.9 Dokumentdarstellung und Markierung der Treffer 119
4.10 Semantisches Highlighting und Übersetzungsfunktionalität 120
4.11 Architektur des thematischen Archivbrowsers 121
4.12 Phasen der Erstellung eines thematischen Inhaltsverzeichnisses 123

5.1 Schematische Darstellung der Benutzeroberfläche des Browser-Editors . . . 141
5.2 Verhältnis zwischen EFGT-Konzepten, Konzeptkandidaten, Stoppwörtern und sonstigen Ausdrücken in einem im Browser-Editor verfügbaren Korpus 143
5.3 Eine Architektur für den Browser-Editor 148
5.4 Interner Aufbau des Indexierers 150
5.5 Vorgänge beim inkrementellen Indexieren 151

6.1 Einsatz der entwickelten Programme im Lebenszyklus eines EFGT-Netzes (vgl. Abb. 2.5, S. 49) 156

Tabellenverzeichnis

1.1 Von Fernández-López und Gómez-Pérez (vgl. Fernández-López and Gómez-Pérez, 2002) untersuchte Methodologien 16
1.2 Einige aktuellere Methodologien zum Aufbau von Ontologien 18

2.1 Thematische Topkategorien des CoGE-Netzes 42

3.1 Geographische Daten bezüglich der Schweiz 69
3.2 Interpretation der Fälle beim Abgleich generierter Einträge 72

5.1 Gegenüberstellung von Aufgaben bei der *Maintenance* bzw. bei der Adaption von EFGT-Netzen 137

Kapitel 1

Ontologien und *Ontology Engineering*

Die vorliegende Arbeit soll Beiträge zur Verbesserung der Vorgehensweise beim Aufbau von EFGT-Netzen liefern, eine spezielle Art von Ontologien. Dieses Kapitel gibt einen allgemeinen Überblick über Ontologien sowie über Vorgehensweisen für deren Entwicklung, sogenannte Methodologien.

Dieser Überblick dient als Vorbereitung für die spätere Definition konkreter Ziele der Arbeit in Kapitel 2, in dem die Entwicklung von EFGT-Netzen näher betrachtet wird. Da EFGT-Netze als Ressource für Anwendungen in der Computerlinguistik und *natural language processing* (NLP) eingesetzt werden, wird in diesem Kapitel außerdem auf die Rolle von Ontologien in diesen Bereichen näher eingegangen.

1.1 Ontologien

Ontologien spielen heutzutage eine wichtige Rolle in intelligenten, wissensbasierten Systemen und sind in vielen Anwendungsbereichen anzutreffen, die fast jeder Computernutzer kennt: E-Commerce, Information Retrieval, Expertensysteme, Bibliothekssysteme, Natural Language Processing, Bioinformatik, usw. In den Anfängen der Entwicklung von solchen intelligenten Systemen musste für jede neue Anwendung das Wissen über den jeweiligen Anwendungsbereich zusammengetragen und formalisiert werden, sodass Ressourcen mit jeweils einem starken Bezug auf eine bestimmte Anwendung angelegt wurden. Da das Zusammentragen von Wissen ein sehr aufwändiger Prozess ist, entstand daraufhin im Bereich der Künstlichen Intelligenz zunächst die Vision, nach der neue Systeme durch Rückgriff auf bereits vorhandene, wiederverwendbare Komponenten entstehen würden (vgl. Neches et al., 1991). Systementwickler sollten sich demnach darauf konzentrieren, einerseits das notwendige Wissen über den Anwendungsbereich zu modellieren und andererseits allgemeine Problemlösungsmethoden und Deduktionsmechanismen zur Automatisierung von Aufgaben und Prozessen zu entwickeln. Neue Systeme würden dann dadurch entstehen, dass man das notwendige Wissen über den spezifischen Anwendungsbereich in einer beschreibenden

Form (= "deklarativ") mit allgemeinen Lösungs- und Deduktionsmechanismen kombinieren würde. Systeme könnten mit anderen, bestehenden Systemen interagieren und auf deren Ergebnissen aufbauen, sodass sie immer umfassender werden könnten und sich durch das Teilen des deklarativen Wissens und der internen Logik insgesamt kostengünstiger bauen ließen.

Die heutige Praxis strebt nach der Umsetzung dieser Vision. Hierbei wird das notwendige Wissen über den Anwendungsbereich in Form von Ontologien deklarativ bereitgestellt: Ontologien liefern das *Vokabular*, das für den jeweiligen Anwendungsbereich oder die Domäne relevant ist. Mit diesem Vokabular wird in der betrachteten Domäne etwa auf Objekte, Arten von Objekten, Begriffe sowie Beziehungen zwischen Objekten, usw. Bezug genommen. Dabei wird von den Details eines bestimmten Systems abstrahiert und der Fokus darauf gelegt, das Charakteristische des Wissensbereiches zu identifizieren und zu erfassen, sodass eine Wiederverwendung innerhalb verschiedener Anwendungen möglich wird. Durch die explizite Modellierung der einzelnen Elemente des Vokabulars mittels einer *Ontologiesprache* wird ihnen eine formale Bedeutung gegeben, die die maschinelle Verarbeitung ermöglicht. Die Anbindung der Ontologie an eine spezifische Anwendung kann dann einfach über einen Interpreten der Ontologiesprache erfolgen.

In Hinsicht auf die Automatisierung von Prozessen soll mit der Implementierung von Semantic Web Services (SWS) für wiederverwendbare Dienste gesorgt werden, die sich zu neuen Anwendungen zusammenstellen lassen. Dennoch ist eine Situation, in der neue Systeme durch Konfiguration und Vernetzung mit anderen Systemen entstehen, bei weitem nicht erreicht. Gründe dafür sind Kosten für das Re-Engineering nach dem neuen Paradigma von bereits bestehenden Anwendungen, die langsame Einigung auf Standards und der Aufwand, den die Entwicklung von Ontologien nach wie vor darstellt.

Die Trennung von Systemlogik und deklarativem Wissen ist an sich ein gutes Designprinzip, das die heutige Verbreitung von Ontologien in der Softwareentwicklung und in anderen Bereichen nur zum Teil erklärt. Ein wesentlicher Faktor ist hingegen das Aufkommen des Internets als gesamtgesellschaftliches Informationsmedium. Die Masse an Information, die dieses von Menschen für Menschen, auf der natürlichen Sprache aufsetzende Medium bereitstellt und oft den einzelnen Benutzer überfordert, hat die Notwendigkeit gezeigt, bestimmte wissensbasierte Aufgaben stärker zu automatisieren. Das *Semantic Web* soll nach Berners-Lee (vgl. Berners-Lee, 1999) eine Erweiterung des heutigen Internets sein, in der Information mit einer wohldefinierten, vom Computer verarbeitbaren Form bereitgestellt wird, sodass sowohl Menschen als auch Computersysteme besser miteinander kooperieren können. Dies soll wie in der ursprünglichen Vision durch die Vernetzung von geteilten, wissensbasierten Komponenten ermöglicht werden, in denen Ontologien das statische, domänenspezifische Wissen stellen. Wiederverwendbare Deduktions- und Problemlösungsmethoden werden dann auf dieses Domänenwissen angewendet, um wissensbasierte Aufgaben zu erledigen. Die Allgegenwärtigkeit und gesellschaftliche Relevanz des Internets bedingt die Aktualität des Semantic Web Paradigma, sodass mittlerweile Standards zur Kodierung von Ontologien vorgeschlagen worden sind und Tools und Methoden zum Aufbau von Ontologien entwickelt werden.

Das gemeinsame, standardisierte Vokabular, das von einer Ontologie bereitgehalten

wird und mit dem Bezug auf eine bestimmte Domäne genommen werden kann, erleichtert vor allem den Austausch und die Wiederverwendung von Wissen über diese Domäne. Nach Grüninger und Lee (Grüninger and Lee, 2002, S. 40) ist ein wichtiger Effekt von Ontologien, die Kommunikation zwischen Menschen, zwischen Computersystemen und zwischen Computersystemen und Menschen zu verbessern. Daraus ergibt sich ein breites Anwendungsspektrum. Ontologien können u.a. eingesetzt werden, um

- den Datenaustausch zwischen Computersystemen zu erleichtern (Interoperabilität)
- die Kommunikation zwischen Softwareagenten zu gewährleisten
- Dienste zur Unterstützung von Wissensarbeitern zu entwickeln, etwa zur Informationssuche in Bibliothekssystemen und im Internet, in Expertensystemen
- die Semantik von strukturierten und semistrukturierten Daten zu kodieren, sodass die einheitliche Abfrage unterschiedlicher Datenbanken ermöglicht wird und sich Daten übersetzen und in andere Formate exportieren lassen
- die Vereinheitlichung und Übersetzung zwischen Wissensrepräsentationsformalismen zu ermöglichen
- Wissensressourcen der Öffentlichkeit zur Verfügung zu stellen
- wissenschaftliche Daten, etwa in der Medizin oder Molekularbiologie, standardisiert zu kodieren und zu annotieren
- Theorien abzubilden
- im Software Engineering die Spezifikation zu erleichtern, Konsistenzprüfungen durchzuführen und die Verlässlichkeit von Programmen zu verbessern (Uschold and Grüninger, 1996, S. 3)
- die Wiederverwendung von Softwarekomponenten zu erleichtern

Aktuell spielen Ontologien eine wichtige Rolle in Disziplinen wie Knowledge Management, Informatik (Semantic Web, Agentensysteme, Webservices), Bioinformatik, E-Commerce.

Als Zusammenfassung und Orientierung für die nächsten Abschnitte sei an dieser Stelle die Definition einer Ontologie von Borst (Borst, 1997, S. 12) wiedergegeben:

> *An ontology is a formal specification of a shared conceptualization.*

Diese Definition erweitert die von Gruber (vgl. Gruber, 1993), der den Begriff einer Ontologie in der Informatik geprägt hat, um das Wort *shared* und fasst somit die angesprochenen Aspekte einer Ontologie in kompakter Form zusammen. Diese werden in den nächsten Abschnitten behandelt. In Abschnitt 1.2, *Ontologien als Formalismus*, geht es um die Mittel, die für die formale Spezifikation einer Ontologie eingesetzt werden. Wie man zu einer

geteilten Konzeptualisierung einer Domäne gelangt und sich in der Praxis Ontologien entwickeln lassen ist das Thema des *Ontology Engineering*, Abschnitt 1.3. Darüber hinaus wird in Abschnitt 1.4 auf die Rolle von Ontologien im *natural language processing* (NLP) eingegangen, da EFGT-Netze vorrangig zum Einsatz in Anwendungen aus diesem Bereich eingesetzt werden.

1.2 Ontologien als Formalismus: Ontologiesprachen

Ein wichtiger Aspekt von Ontologien ist, dass es sich um *formale Spezifikationen* handelt. In einer solchen Spezifikation werden die Begriffe, Objekte, Relationen, usw., die das Vokabular der Ontologie ausmachen, einerseits explizit benannt und andererseits deren Bedeutung präzise definiert. Für diese Definition wird eben auf eine *formale* Sprache zurückgegriffen, die Ontologiesprache. In der Praxis heißt das, dass eine von einer Vielzahl von logikbasierten Sprachen verwendet wird, die sich als Ontologiesprachen etabliert haben, und die bekannte, günstige formale Eigenschaften hat. Damit kann dem Vokabular eine formale Bedeutung (Semantik) gegeben werden. Somit entspricht eine Ontologie einem mathematischen Modell, das sich als Struktur – etwa als Baum oder Netz – repräsentieren lässt und mit dem das abstrakte Modell, mit dem über die Wissensdomäne gedacht wird, eingefangen wird. Dank der formalen Eigenschaften der Ontologiesprache lässt sich die Ontologie in ihrer wohldefinierten Bedeutung maschinell verarbeiten. Unter anderem lassen sich dann verschiedene Konsistenztests durchführen und bestimmte Fakten aus der Ontologie ableiten (*automatische Inferenz*).

Ontologiesprachen unterscheiden sich in den Ausdrucksmitteln, die sie für die Angabe dieser Spezifikation bereitstellen. Allen Ontologiesprachen gemeinsam ist die Möglichkeit, ein Vokabular von Bezeichnungen für verschiedene Arten von Objekten und Relationen zwischen Objekten in dem jeweiligem Wissenbereich zu definieren. Jedes der Elemente dieses Grundvokabulars lässt sich mit Hilfe eines Systems eindeutiger Identifikatoren von den anderen, im anvisierten Wissensbereich unterschiedenen Objekten auseinanderhalten. Arten von Objekten werden als *Klasse* oder *Konzept* bezeichnet und stellen aus logischer Sicht Mengen dar, die in einer Ontologie typischerweise über die Teilklassenrelation oder auch *is-a*-Relation in einer *Taxonomie*, einer hierarchischen Struktur, organisiert sind. Die Bedeutung einer Klasse kann oft durch die Angabe von Merkmalen (*properties*, aber auch *slots* oder *roles*) weiter präzisiert werden, die Relationen zu anderen Klassen, zu einzelnen Objekten oder auch Datenwerten bezeichnen. Je nach Ontologiesprache lassen sich die Eigenschaften der *properties* mit Hilfe von sog. *facets* oder *role restrictions* weiter einschränken, etwa in ihrer Multiplizität oder ihren Eigenschaften (Transitivität, usw.). Manche Sprachen bieten die Möglichkeit, eine Teilrelation-Relation zu spezifizieren und dadurch eine Hierarchie von Relationen definieren zu können. Bei mächtigeren Ontologiesprachen besteht die Möglichkeit, über *Konnektoren* und *Quantoren* neue Konzepte aufbauend auf bestehenden Konzepten und Relationen zu definieren und auf diese Weise das Vokabular zu erweitern. Mit manchen Ontologiesprachen kann darüber hinaus die Bedeutung der Konzepte und Relationen in der Ontologie durch die Angabe allgemeiner *Axiome* näher

eingeschränkt werden.

Zusätzlich zur Ontologie kann eine Auflistung von Objekten, sog. *Instanzen*, zu den verschiedenen Klassen der Ontologie angegeben werden. Typischerweise übersteigt die Anzahl der Instanzen bei weitem die Anzahl der Klassen in der Ontologie, bei Angabe von Instanzen zu einer Ontologie spricht man auch von einer *Wissensbasis*.

Zu den wichtigen formalen Eigenschaften von Ontologiesprachen gehören *Entscheidbarkeit* und *Komplexität*. Diese begrenzen die Ausdrucksmittel, die eine Ontologiesprache bereitstellen darf, d.h deren *Ausdrucksstärke*, falls der praktische Nutzen der Sprache erhalten werden soll. Entscheidbarkeit behandelt die Frage, ob sich für jeden Ausdruck der Ontologiespezifikation feststellen lässt, ob sie zwangsläufig gilt, nicht gilt oder gelten kann. Ein System, das dies überprüft (*Deduktionssystem*) sollte korrekt und vollständig sein, d.h. nur gültige Schlüsse ziehen bzw. für jeden mittels der Sprache gebildeten Ausdruck einen Schluss ziehen können. Dies wird oft durch die Komplexität der Algorithmen bedingt, die dafür benötigt werden. Komplexität bezieht sich auf die Zeit und den Speicherplatz, die im Verhältnis zur Größe des Problems benötigt werden, um dieses zu lösen. Eine *polynomielle* Komplexität ist im Allgemeinen akzeptabel, während eine *exponentielle* als nicht mehr nutzbar gilt, da für realistische Aufgaben zu lange gebraucht wird. Beispielsweise machen die Ausdrucksmittel, die die Prädikatenlogik bereitstellt, sie unentscheidbar. An Ausdrucksstärke zu gewinnen, aber innerhalb der Entscheidbarkeit zu bleiben, ist eine Motivation für die Entwicklung verschiedener Logiken und Ontologiesprachen. Da lautet die Forschungsfrage oft, wie Algorithmen zu finden sind, die "im Normalfall" schnell eine Lösung versprechen und nur in einzelnen Fällen einen exponentiellen Aufwand besitzen.

Wie bereits angedeutet, haben die Fragen der Entscheidbarkeit und der Verfügbarkeit eines Deduktionssystems bei der Entwicklung einer Ontologie eine praktische Relevanz. In der Phase, in der eine Ontologie entwickelt wird, ist es nützlich, verschiedene Tests an der Ontologie durchzuführen. Dazu gehören Tests wie die Konsistenz einer Klasse, d.h. zu testen, ob sie ihrer Definition nach tatsächlich Objekte enthalten kann, sich die Hierarchie der Klassen (Taxonomie) berechnen zu lassen oder die für eine Instanz inferierten Typen zu ermitteln, d.h. die Zugehörigkeit der Instanz zu den verschiedenen Klassen zu berechnen. Um diese Tests durchzuführen ist eine Inferenzmaschine notwendig.

In Abb. 1.1 sind Ontologiesprachen nach ihrer Ausdrucksstärke in der sog. Semantischen Treppe geordnet. Je stärker die Ausdrucksstärke, desto komplexere Strukturen können mit der Ontologiesprache definiert werden, desto "semantisch reichhaltiger" können die damit spezifizierten Ontologien sein und desto größer ist der Vorteil einer vorhandenen Inferenzmaschine. Nicht immer ist eine hohe Ausdrucksstärke notwendig. So stellen einfachere, sog. *light-weight* Ontologien eine Taxonomie dar, deren Klassen über eine geringe Anzahl von *properties* zusätzlich verbunden sind, sowie einzelne zusätzliche Relationen. Die Spezifikation von *heavy-weight* Ontologien benötigt dagegen erweiterte Ausdrucksmittel, um etwa *role restrictions* zu verwenden oder Eigenschaften von Relationen mittels Axiomen festzulegen.

Es gibt eine Reihe von Formalismen, die von vornherein als Ontologiesprache konzipiert wurden. Im Bereich der Künstlichen Intelligenz sind in den neunziger Jahren verschiedene

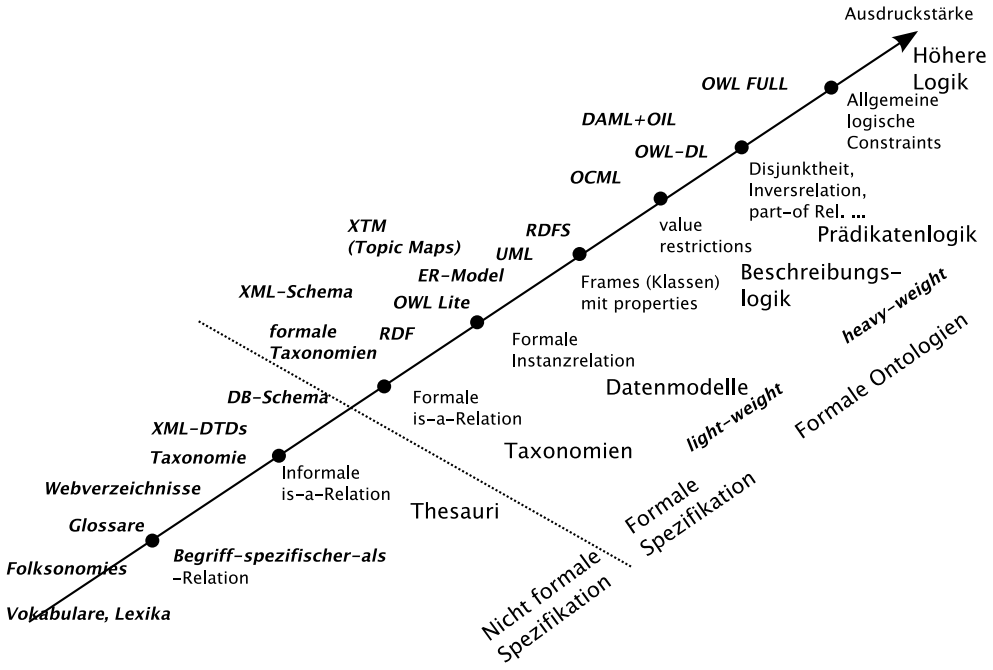


Abbildung 1.1: Ontologiesprachen auf der Semantischen Treppe. (Vgl. Uschold and Grüninger, 2004; Lassila and McGuinness, 2001)

Ontologiesprachen entstanden, wie z.B. Ontolingua[1], OCML, usw. (vgl. Abb. 1.2), ohne dass es sich daraus ein Standard herausbildete. Mit dem Aufkommen des Semantic Web haben die von W3C Consortium propagierten Formalismen wie RDF und OWL, die auf Websprachen aufsetzen, eine große Verbreitung erreicht. Diese und der zunehmende Grad der Standardisierung sind wichtige Voraussetzungen dafür, dass eine Ontologiesprache als Datenaustauschformat dienen kann und das Versprechen der Wiederverwendung eingelöst werden kann. Abb. 1.2 stellt die Evolution der verschiedenen Ontologiesprachen dar. Eine Einordnung des EFGT-Formalismus' als Ontologiesprache erfolgt später in Abschnitt 2.2.1.

Eine wichtige Unterscheidung ist die zwischen der Syntax, die von der Ontologiesprache definiert wird, und dem mathematischen Formalismus (oft eine bestimmte Logik), der die Semantik für die Ausdrücke in dieser Syntax bereitstellt. So hat OWL eine auf XML aufbauende Syntax, es werden aber verschiedene Teilsprachen von OWL unterschieden in Abhängigkeit davon, welcher Formalismus davon abgedeckt wird. Jede Teilsprache übernimmt die formalen Eigenschaften des entsprechenden Formalismus', eben seine Ausdrucksstärke. OWL-Lite entspricht der Aussagenlogik, während OWL DL einer bestimmten Beschreibungslogik, nämlich SHIOQ, gleichzusetzen ist. Die von der gesamten OWL Spezifikation umfasste Syntax (OWL Full) entspricht einer Logik, die nicht mehr entscheidbar ist. Ähnlich umfasst KIF einen Bereich der Prädikatenlogik, wobei die konkrete Syntax

[1]Literaturverweise zu allen in diesem Abschnitt erwähnten Ontologiesprachen können Gómez-Pérez et al., 2003, Kap. 4 entnommen werden.

sich an der Programmiersprache LISP orientiert.

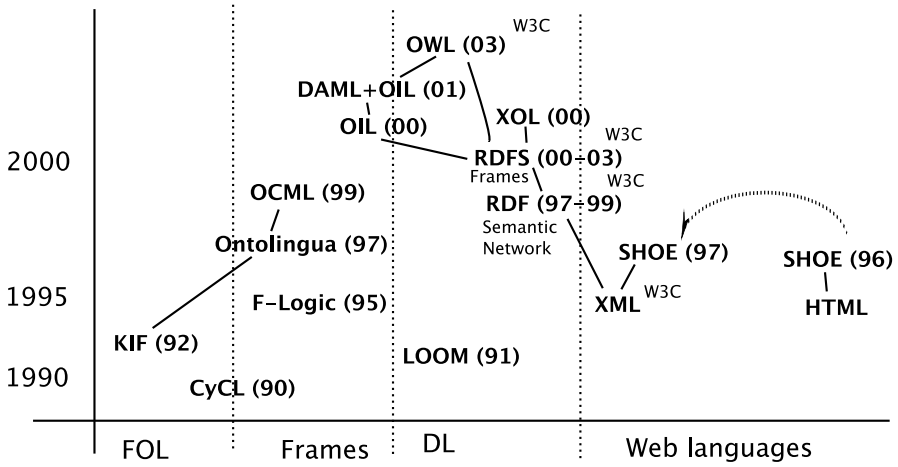


Abbildung 1.2: Evolution von Ontologiesprachen (Gómez-Pérez et al., 2003, S. 200 ff). Abkürzungen: FOL: *first order logic* (Prädikatenlogik); DL: *description logics* (Beschreibungslogiken); Frames: Frameformalismen; W3C: Empfehlung des W3C. Verbindungen zwischen Ontologiesprachen deuten aufeinander aufbauende Spezifikationen bei Websprachen bzw. Abwandlungen bei FOL- und Framesprachen an. Die Trennung zwischen den zugrunde liegenden Formalismen in der unteren Achse ist nur grob.

In der Informatik und Künstlichen Intelligenz sind verschiedene Daten- und Wissensmodelle vorhanden, die nicht primär als Ontologiesprachen entwickelt wurden, die zumindest einen Teil der Grundelemente einer Ontologie (Klassen, Relationen zwischen Klassen, Instanzen, usw.) zur Verfügung stellen und einen formalen Charakter haben, sodass sie sich nach obiger Definition dafür eignen würden, Ontologien zu spezifizieren. Ein Beispiel hiervon wäre das Entity-Relationship-Modell. Die Ausdrucksstärke dieser Formalismen lässt jedoch meistens nur die Definition von *light-weight*-Ontologien zu.

Darüber hinaus lassen sich mit informatischen Mitteln prinzipiell Strukturen darstellen, die sich als Relationen zwischen Elementen eines Vokabulars auffassen lassen. Auch wenn das Ziel sein kann, so etwa bei Webverzeichnissen oder Produktkatalogen, einen Wissensbereich zu modellieren, der sich sogar über geeignete Schnittstellen in andere Anwendungen einbinden lässt, wird hier das Kriterium einer formalen, expliziten Spezifikation verletzt. Ein Teil dieser zusätzlichen Mittel ist ebenfalls auf Bild 1.1 eingetragen.

Bei der Wahl einer Ontologiesprache für ein bestimmtes Projekt sind technisch-technologische Aspekte sowie die Mächtigkeit der Sprache zu berücksichtigen. Was hierbei sinnvolle Kriterien wären, wird in der Literatur wenig besprochen. Grund hierfür ist eventuell die beobachtbare, zunehmende Standardisierung in dem Bereich, seitdem das Semantic Web ausgerufen wurde. In Anlehnung an Gómez-Pérez et al. (Gómez-Pérez et al., 2003, S. 202 ff) seien hier folgende mögliche Kriterien erwähnt:

- Welche Ausdrucksstärke ist notwendig, um die Domäne sinnvoll zu modellieren? Ist Inferenz nötig bzw. sind Inferenzmaschinen für die anvisierte Sprache verfügbar?
- Ist eine spezifische Ontologie-Entwicklungsumgebung verfügbar, etwa ein Editor für die Sprache?
- Welche Anbindungsmöglichkeiten an die Anwendung ergeben sich aus der Wahl, etwa über einen für die Sprache geeigneten Interpreten? Lässt sich ein solcher Interpreter leicht austauschen?
- Ist die Integration von Weblanguages notwendig für die spätere Verwendung?
- Ist eine (verlustfreie) Übersetzung von eventuell vorhandenen Daten in das anvisierte Ontologieformat möglich?
- Sind Personen verfügbar, die die Ontologiesprache verstehen und in der Lage sind, sie anzuwenden? Der Schritt über Deduktionsmechanismen bei ausdrucksstarken Ontologiesprachen ist oft nicht unmittelbar nachvollziehbar. In vielen Fällen werden deshalb Formalismen bevorzugt, bei denen man die Struktur direkt angeben kann.

Beschäftigt man sich mit der Beantwortung dieser Fragen, ist man bereits mitten im *Ontology Engineering* angekommen.

1.3 Entwicklung von Ontologien: *Ontology Engineering*

Ontologien sind formale Spezifikationen einer geteilten Konzeptualisierung eines Wissensbereichs. Aber wie gelangt man zu einer solchen, d.h. zu einem abstrakten Modell mit allen relevanten Begriffen und Relationen? Dass diese Konzeptualisierung außerdem noch *geteilt* sein soll, bedeutet, dass sie den Konsens der Vorstellungen über den Bereich treffen soll, um die Ontologie in unterschiedlichen Systemen und Szenarien erfolgreich einsetzen zu können. Wie lässt sich ein solcher Konsens erreichen?

Mit diesen praktischen Aspekten der Entwicklung von Ontologien beschäftigt sich das Feld des *Ontology Engineering*. Weitere damit verbundenen Fragen sind:

- Was ist eine gute Ontologie und wie kann man Ontologien bewerten?
- Welche Arten von Ontologien gibt es? Wie sehen typische Ontologie-Entwicklungsprojekte aus?
- Wie kann man Ontologien entwickeln? Was sind dabei gute Vorgehensweisen und Methoden, auch im Sinne von Effizienz und Kostengünstigkeit?
- Nach welchen Kriterien soll eine Ontologiesprache für die Formalisierung gewählt werden? Welcher Grad der Formalisierung soll realistisch angestrebt werden?

- Wie lässt sich das notwendige Wissen über den zu modellierenden Bereich akquirieren?
- Welche Technologien und Systeme sind verfügbar und unterstützen den Entwicklungsprozess?

Seit seinen Anfängen wird das Feld des *Ontology Engineering* vom Software Engineering beeinflusst und hat eine ähnliche Entwicklung insofern, dass verschiedene Modelle vorgeschlagen wurden.

Antworten zu den Fragen oben werden im Rahmen von den in der Literatur vorgeschlagenen Methodologien zur Entwicklung von Ontologien gegeben. Auf diese Methodologien wird hier eingegangen, nachdem typische Ontologieprojekte vorgestellt werden. Welche technischen Möglichkeiten es zur Unterstützung des gesamten Entwicklungsprozesses gibt, wird in Abschnitt 1.3.3 beschrieben. Wie daraus resultierende Ontologien evaluiert werden können wird in Abschnitt 1.3.4 beschrieben.

1.3.1 Arten von Ontologien

Neben der angesprochenen, eher pragmatischen Unterscheidung zwischen *light-weight-* und *heavy-weigth*-Ontologie kann man sich fragen, wie eine typische Ontologie aussieht bzw. welche Arten von Ontologien entwickelt werden. Hepp et al. (Hepp et al., 2008, S. 8-9) schlagen sechs Parameter zur Charakterisierung eines Ontologie-Projektes vor, die Einfluss auf den Entwicklungsprozess der Ontologie nehmen:

- Die Ausdrucksstärke des eingesetzten Ontologieformalismus'
- Die Größe der adressierten Nutzergemeinschaft. Das Erreichen von Konsens wird hiervon bedingt.
- Der Grad der Dynamik auf der Konzeptualisierungsebene, d.h wie oft neue Konzepte und Änderungen bei konzeptuellen Elementen stattfinden. Wie detailliert eine Ontologie sein kann und wie leicht sie sich warten lässt, hängt mit diesem Parameter zusammen.
- Die Anzahl der konzeptuellen Elemente im anvisierten Wissensbereich. Die Visualisierung einer Ontologie ist umso schwieriger, je mehr Elemente sie umfasst. Ebenfalls müssen entsprechend effiziente Deduktionsmechanismen eingesetzt werden. Kleinere Ontologien werden schneller angenommen.
- Der Grad der Subjektivität im jeweiligen Wissensbereich. Für sehr subjektive Bereiche lässt sich nur eine ungenaue Spezifikation der Ontologie angeben. Eventuell muss ein spezifisches Verfahren zur Konsensbildung eingesetzt werden.
- Die Durchschnittsgröße der Spezifikation der einzelnen konzeptuellen Elemente. Dieser Faktor ist für den Aufwand der Kodierung oder die Leichtigkeit, mit der eine Spezifikation angenommen wird, ausschlaggebend.

Auf der Grundlage dieser Parameter ließe sich beispielsweise ein Strahlendiagramm erstellen, in dem Profile für unterschiedliche Ontologien eingetragen werden. In der Literatur finden sich vielmehr Ontologieklassifikationen, so etwa nach Anwendungsbereichen, nach der Mächtigkeit des eingesetzten Formalismus', nach der modellierten Domäne, dem Grad der Formalisierung, usw. Ohne dabei ein spezifisches Kriterium zu verfolgen, werden in der folgenden Auflistung einzelne, oft erwähnte Arten von Ontologien aufgeführt:

***Upper-Level*-Ontologien:** *Upper-Level-* oder auch *Top-Level*-Ontologien modellieren sehr generische, grundlegende Konzepte wie *Zeit, Raum, Entität, Materie*, usw., die sich letztendlich in jedem spezifischen Wissensbereich wiederfinden lassen. *Upper-Level*-Ontologien dienen als Grundlage für die Modellierung von spezifischen Domänen. Domänenontologien können auf eine *Upper-Level*-Ontologie verweisen. Beispiele sind die *Suggested Upper Merged Ontology* (SUMO; vgl. Teknowledge Corporation, 2000) oder die *General Formal Ontology* (GFO; vgl. Herre et al., 2007)

Domänenontologien: Die Mehrzahl der verfügbaren Ontologien machen Domänenontologien aus. Sie modellieren einen bestimmten Teil der Realität und stellen oft das Wissen einer Gruppe von Fachexperten dar. Wissensbereiche, in denen Ontologien eine breite Verwendung finden, sind bspw. Medizin, Genetik, Molekularbiologie. Im Sinne ihrer Formalisierung sind Domänenontologien eine heterogene Gruppe. Dies hat oft historische Gründe, da man auf bereits vorhandene Daten wie etwa Taxonomien oder Terminologien zurückgreift. So wurde etwa bei der Herausgabe einer Ontologie wie das Unified Medical Language System (UMLS; vgl. U.S. National Library of Medicine, 2006) von an die hundert bereits aktiv gebrauchten Vokabularien ausgegangen, um sie mittels einer leichtgewichtigen semantischen Schicht, das sog. Semantic Network, zu integrieren. Bezüglich der Modellierung stellt das Semantic Network einen Kompromiss dar, um die Vielfalt der eingebundenen Vokabularien überhaupt berücksichtigen zu können. Andere Domänenontologien wurden bereits von Anfang an als Ontologie konzipiert und sind in einer Ontologiesprache frei verfügbar, wie etwa das *Foundational Model of Anatomy* (vgl. Structural Informatics Group, 2009), das als OWL-Datei erhältlich ist. Im Fall der *Gene Ontology* (GO; vgl. The Gene Ontology Consortium, 2000) bezieht sich die Bezeichnung Ontologie vor allem auf die Tatsache, dass es sich dabei um ein allgemein akzeptiertes, strukturiertes Vokabular handelt, das mit Hilfe der entsprechenden fachlichen Gemeinschaft entwickelt wurde. Die Bestrebung einer expliziten Formalisierung dessen Bedeutung ist erst entstanden, als die GO verbreitet Anwendung gefunden hatte. In anderen Wissensbereichen fällt die Entstehungsgeschichte von Ontologien ähnlich aus. Inzwischen sind in einzelnen Bereichen Ontologiebibliotheken erstellt worden; so stellt bspw. das *Open Biomedical Ontologies (OBO) Foundry* (vgl. Smith et al., 2007) eine Vielzahl von Ontologien aus dem Bereich der Biomedizin zur Verfügung. Nicht domänenspezifische Bibliotheken und Suchmaschinen für Domänenontologien sind u. a. OntoSelect (vgl. Buitelaar et al., 2009) und Swoogle (vgl. Li et al., 2004).

Linguistische Ontologien: Unter den linguistischen Ontologien kann man zwischen solchen unterscheiden, die auf linguistische Theorien Bezug nehmen und bspw. für die

Annotation von Corpora verwendet werden, und solchen, die wie ein Lexikon den Wortschatz einer natürlichen Sprache erfassen, dessen Bedeutung aber mit Hilfe von Konzepten und semantischen Beziehungen zumindest grob formalisieren. Zu der ersten Gruppe zählen Ontologien wie das *Generalized Upper Model* (vgl. Bateman et al., 1995), das den Charakter einer linguistisch motivierten Top-Level-Ontologie hat, oder GOLD (vgl. Farrar and Langendoen, 2003) zur Annotation sprachlicher Daten. Das prominenteste Beispiel der Gruppe der linguistischen Ressourcen ist WordNet (vgl. Fellbaum, 1998). In diese Gruppe lassen sich ebenfalls EFGT-Netze einordnen. Auf die Rolle von linguistischen Ontologien im *natural language processsing* (NLP) wird in Abschnitt 1.4 näher eingegangen.

Themenverzeichnisse, Produktkataloge: Bei manchen Internetanwendungen dienen Strukturen, die einer Ontologie ähneln, dazu, den Zugang auf bestimmte Informationen zu ermöglichen. Im Information Retrieval Bereich etwa ordnen Webverzeichnisse Internetseiten in eine nicht formale Hierarchie von Themen ein und erleichtern damit die Navigation im Netz; beim E-Commerce werden oft hierarchisch organisierte Produktkataloge eingesetzt, um die Informationen zum Sortiment für den potenziellen Käufer zugänglich zu machen. Auch wenn diese Strukturen teilweise umfangreiche Modellierungen eines Wissensbereichs darstellen, ist die Bezeichnung dieser Strukturen als Ontologien nicht gerechtfertigt, da sie in den meisten Fällen nicht in einer expliziten, formalen Spezifikation vorliegen. Das von Internetnutzern gepflegte Verzeichnis Open Directory Project (vgl. Netscape, 2009), das von der Firma Netscape betrieben wird, kann im RDF-Format heruntergeladen werden.

Anwendungsontologien bzw. informatische Ontologien: Das von Anwendungsontologien umfasste Vokabular bezieht sich auf Prozesse und Zustände einer bestimmten Anwendung, etwa ein Workflowsystem; in einer spezifischen Anwendung kann eine eingesetzte Domänen- durch eine Anwendungsontologie erweitert werden. In verschiedenen Bereichen der Informatik sind Ontologien ein Hilfsmittel beim Aufbau der Infrastruktur, etwa bei der Schema-Integration oder bei Semantic Web Services. Diese Art von Ontologien modellieren keine allgemeine Domäne, sondern nehmen einen starken Bezug auf die Komponenten und die Anwendung, innerhalb derer sie vermitteln. Ontologien lassen sich auch in verschiedenen Phasen des Software Engineering Prozesses einsetzen, so bei der Anforderungsanalyse, bei der Modellierung der Software und der Anwendungsdomäne, zur Dokumentation von Interfaces, zur Modellierung von Middleware, als Grundlage für Business Rules, usw.

Wissensrepräsentationsontologien: Die unterschiedlichen Primitiva, die in einer spezifischen Ontologiesprache verfügbar sind, können ihrerseits als Konzepte aufgefasst werden. So können beispielsweise die vom RDF-Schema bereitgehaltenen Sprachelemente, mit dem der Aufbau von Daten in RDF beschrieben werden kann, als Wissensrepräsentationsontologie betrachtet werden.

***Task*-Ontologien:** Diese Art von Ontologien stellt Vokabular zur Beschreibung einer ge-

nerischen Aktivität oder Aufgabe, z.b. diagnostizieren oder die Definition von Ablaufplänen von Anwendungen (vgl. Rajpathak et al., 2001).

1.3.2 Methodologien zur Entwicklung von Ontologien

Mit der Entwicklung einer Ontologie sind verschiedene, vielfältige Aktivitäten verbunden. Aus diesem Grund ist im Ontology Engineering meistens die Rede (in Anlehnung an den englischen Sprachgebrauch) von *Methodologien* zur Entwicklung von Ontologien im Sinne einer Sammlung von Methoden. Die im deutschen Sprachgebrauch übliche Auffassung von Methodologie als Lehre von Methoden, die diese systematisch miteinander vergleicht und daraus Richtlinien für die Praxis zu identifizieren versucht, ist damit in der Regel nicht intendiert. Dennoch wird im Folgenden die Bezeichnung weitergeführt, um zwischen einzelnen Methoden und vorgeschlagenen Methodensammlungen unterscheiden zu können.

In der Literatur zum Ontology Engineering ist eine Vielzahl von Methodologien beschrieben worden, jedoch hat sich keine davon als *die* Methodologie etabliert oder haben sich Paradigmen wie im Software Engineering herausgebildet, wie etwa die Agile Softwareentwicklung (vgl. Fowler, 2005) oder das Rational Unified Process (vgl. Jacobson et al., 1999). Wie Bontas und Tempich (vgl. Bontas and Tempich, 2006) anmerken, zielen im Ontology Engineering die Mehrzahl der Arbeiten, die eine bestimmte Methodologie vorschlagen, darauf ab, deren Durchführbarkeit zu zeigen oder die Anwendung eines bestimmten Tools im Rahmen der Ontologieentwicklung zu demonstrieren. Dagegen ist die Anzahl der Arbeiten gering, die eine bereits beschriebene Methodologie aufgreifen und sie innerhalb eines konkreten Ontologieprojektes anwenden. Das bedeutet, dass die meisten Methodologien nur in dem Rahmen getestet wurden, für den sie entworfen wurden und, abgesehen von Einzelfällen, vom selben Kreis von Personen, die an der Entwicklung beteiligt waren. Dementsprechend ist wenig bekannt über die Übertragbarkeit vorhandener Methodologien auf andere Bedingungen und fehlen Beschreibungen, die die unterschiedlichen Aktivitäten und Prozesse beim Ontology Engineering *im Allgemeinen* operationalisieren.

Das Ziel der vorliegenden Arbeit ist weder die Entwicklung einer allgemeinen Methodologie oder einen systematischen Vergleich durchzuführen – eine echte Methodologie nach dem deutschen Sprachgebrauch – noch Vorschläge zur Operationalisierung einzelner Aktivitäten im Ontology Engineering zu machen, sondern, ausgehend von der tatsächlichen Vorgehensweise bei der Entwicklung und Pflege von EFGT-Netzen, die existierende technische Arbeitsumgebung des Ontologieentwicklers zu erweitern und somit einen Beitrag dazu zu leisten, den gesamten Entwicklungsprozess effizienter zu gestalten.

Bevor in Kapitel 2 auf die spezifischen, mit der Entwicklung von EFGT-Netzen verbundenen Aspekte und Probleme eingegangen wird und konkrete Ziele für diese Arbeit ausgearbeitet werden, dient der folgende Überblick über existierende Methodologien dazu, vorab verschiedene Gestaltungsmöglichkeiten bei der Entwicklung von Ontologien und sinnvolle Maßnahmen aufzuzeigen.

In den folgenden Abschnitten werden zunächst die verschiedenen Aktivitäten beschrieben, die eine Methodologie zu berücksichtigen hat und die den sog. Lebenszyklus einer Ontologie ausmachen. Anhand der Studie von Fernández-López und Gómez-Pérez (vgl.

Fernández-López and Gómez-Pérez, 2002) werden anschließend allgemeine Kriterien zum Vergleich von Methodologien vorgestellt. Vor dem Hintergrund der Schlussfolgerungen, die die Autoren dieser Studie mit Hilfe dieser Kriterien aus der Betrachtung damaliger Methodologien ziehen konnten, werden neuere Entwicklungen aufgezeigt, wie sie sich aus aktuelleren Arbeiten entnehmen lassen. Auf die Rolle einer Endanwendung der Ontologie in aktuellen Methodologien und auf die Bildung von Konsens über Ontologien in kollaborativen Szenarien wird in jeweils eigenen Abschnitten eingegangen ohne Anspruch, systematisch vergleichen zu wollen. Diese eher theoretischen Betrachtungen werden den Ergebnissen der empirischen Studie von Bontas und Tempich (vgl. Bontas and Tempich, 2006) gegenübergestellt, um das Bild des aktuellen Standes von Methodologien im Ontology Engineering zu vervollständigen.

Aktivitäten im Ontology Engineering und der Lebenszyklus einer Ontologie

Die unterschiedlichen Aktivitäten, die während der Entwicklung von Ontologien stattfinden, wird hier anhand der Beschreibung des *ontology development process* innerhalb der Methodologie METHONTOLOGY (vgl. Fernández-López et al., 1997) dargestellt. Die METHONTOLOGY ist eine der ersten systematischen Methodologien, wobei sie sich noch stark an Methoden des Software Engineering orientiert. So werden darin die verschiedenen Aktivitäten zur Entwicklung einer Ontologie noch in Anlehnung an den IEEE-Standard für den Software-Entwicklungsprozess (vgl. IEEE, 1996) beschrieben und in Gruppen unterteilt.

Die Gruppe der *Management-Aktivitäten* fasst alle Aktivitäten zusammen, die dazu dienen, den gesamten Entwicklungsprozess zu steuern. Dazu gehören etwa die Definition eines Zeitplans für den gesamten Ablauf (*scheduling*) als auch Controlling und Qualitätssicherungsmaßnahmen.

Die sog. *development oriented activities* stellen den eigentlichen Entwicklungsprozess dar. Sie lassen sich in drei Phasen gliedern:

- *pre-development*-Aktivitäten, wie die Durchführung von Machbarkeitsstudien, um zu entscheiden, ob die Entwicklung einer Ontologie bzw. einer ontologiebasierten Anwendung eine gangbare Lösung ist
- die faktische Entwicklung der Ontologie (*development activities*), die wiederum weiter unterteilt werden kann in:
  - die *Analyse der Domäne*, die die Identifikation typischer Einsatzszenarien, die Spezifikation bzgl. der angedachten Anwendung und die Analyse eventuell bereits existierender Lösungen umfasst;
  - der *Konzeptualisierungsphase*, in der ein abstraktes Modell entworfen und formalisiert wird, ggf. unter Berücksichtigung bestehender Modelle, die integriert oder erweitert werden;
  - der *Implementierung* der Ontologie in eine bestimmte Ontologiesprache;

- *post-development*-Aktivitäten fallen mit dem Einsatz der Ontologie im Rahmen von Anwendungen an. Mit *maintenance* wird die Wartung der Ontologie und das Beheben von Fehlern bezeichnet, die sich bei ihrer Verwendung zeigen, wohingegen *(re)use* auf die Integration und den Einsatz im Rahmen einer oder mehrerer Anwendungen sowie die Adaption an neue Anforderungen verweist.

Parallel zur Entwicklung findet eine Reihe zusätzlich unterstützender Aktivitäten statt, sog. *ontology support activities*:

- Wissensakquisition, d.h. Ermittlung der konzeptuellen Elemente, die das abstrakte Modell der Domäne ausmachen, etwa durch Befragung von Domänenexperten oder durch automatisches *ontology learning* auf der Grundlage von verfügbaren Daten;
- Evaluation der Ontologie;
- Wiederverwendung und Integration bestehender Ontologien, entweder, indem sie in die entstehende Ontologie eingearbeitet werden (*ontology merging*) oder auf sie referenziert wird (*ontology alignment*);
- Dokumentation;
- *configuration management*: technisches Management der verschiedenen Entwicklungsstadien bzw. Versionen der Ontologie, Dokumentation der Änderungen, usw..

Charakteristisch für eine bestimmte Methodologie ist der spezifische Ablauf, in dem die einzelnen Aktivitäten durchgeführt werden sollen, der sogenannte *Lebenszyklus einer Ontologie*. Je nach verfolgter Methodologie durchläuft die Ontologie somit unterschiedliche Schritte oder Stadien, die jeweils an bestimmte Aktivitäten gekoppelt sind.

Die Vielfältigkeit der Aktivitäten deuten den potentiellen Aufwand und die Größe der Organisation an, die unter Umständen für die Entwicklung einer Ontologie notwendig sind. Unter den verschiedenen Beteiligten an einem Ontologieentwicklungsprojekt lässt sich zwischen

- den Experten der anvisierten Wissensdomäne, die das zu modellierende Wissen bereitstellen,
- den Ontologieentwicklern (*ontology* oder *knowledge engineers*), deren Hauptqualifikation die Vertrautheit mit dem Ontologieformalismus und der Entwicklungsumgebung ist
- und den Endnutzern der Ontologie im Rahmen einer bestimmten Anwendung

unterscheiden. Manche Methodologien greifen diese Unterscheidung auf und weisen bestimmten Personenkreisen verschiedene Aktivitäten zu oder beteiligen sie an unterschiedlichen Phasen des Ontologielebenszyklus'. In anderen Fällen wird weniger scharf differenziert, so fällt etwa oft die Gruppe der Domänenexperten mit dem Endnutzerkreis zusammen.

Kriterien zum Vergleich von Methodologien

Liegt ein Katalog von Aktivitäten wie oben beschrieben vor, können Methodologien zur Entwicklung von Ontologien ("echte" Methodologie, s.o.) dadurch verglichen werden, dass die darin definierten Aktivitäten und die jeweiligen Vorschläge für deren Durchführung gegenübergestellt werden. Ein solcher Vergleich der *Aktivitäten im Ontologieentwicklungsprozess* ist eins der Kriterien, die in der Studie von Fernández-López und Gómez-Pérez (vgl. Fernández-López and Gómez-Pérez, 2002) eingesetzt wurden, um sieben Methodologien aus den Jahren 1995-2000 zu analysieren und zu bewerten. Als weiteres Kriterium wurde in dieser Studie der Vergleich von dem, was die Autoren unter *Konstruktionsstrategie* zusammenfassen und verschiedenen Aspekten der jeweiligen Methodologie berücksichtigt:

- Angaben zum Lebenszyklus der Ontologie, wobei darunter übliche Modelle aus der Softwareentwicklung verstanden werden:
  - inkrementelle Entwicklung in Versionen (vgl. McCracken and Jackson, 1982): Verschiedene Zustände der Ontologie werden mit Versionsnummern versehen; nötige neue Definitionen und sonstige Änderungen an einer Version der Ontologie werden in eine darauffolgende Version aufgenommen.
  - im *evolving prototypes*-Modell (vgl. Kendall and Kendall, 1995): Änderungen an der Ontologie dürfen nach Bedarf jederzeit vorgenommen werden; die Ontologie stellt einen Prototyp dar, an dem fortwährend gearbeitet wird.
- Strategie bzgl. der Anwendung: Die Entwicklung kann entweder *anwendungsorientiert* sein, d.h. auf den Einsatz in einer bestimmten Anwendung fokussiert, *anwendungsangelehnt*, indem bereits in der Spezifikation mögliche Anwendungen der Ontologie berücksichtigt werden, oder *anwendungsunabhängig*, d.h. losgelöst von der Betrachtung möglicher Anwendungen.
- Verwendung von *core ontologies* als Grundlage für die neue Ontologie: Damit ist sowohl die Verwendung einer Top-Level-Ontologie als auch die Integration bereits vorhandener domänenspezifischer Wissensressourcen gemeint.
- Strategie zur Identifikation von relevantem Vokabular und Aufbau der Taxonomie (vgl. Uschold and King, 1995; Uschold and Grüninger, 1996):
  - *bottom-up*: Zunächst werden die spezifischsten Begriffe der Domäne identifiziert, um anschließend die gliedernde Taxonomie durch schrittweises Generalisieren aufzubauen;
  - *top-down*: Der Aufbau beginnt bei den allgemeinsten Begriffen, die dann sukzessiv weiter spezialisiert werden;
  - *middle-out*: Zunächst sind die relevantesten, frequenten Begriffe zu identifizieren, die dann taxonomisch zueinander in Beziehung gesetzt werden.

Darüber hinaus verwenden Fernández-López und Gómez-Pérez (vgl. Fernández-López and Gómez-Pérez, 2002) als Vergleichskriterium die Möglichkeiten der *technologischen Unterstützung*. Diese bezieht sich im Wesentlichen auf die verfügbaren Tools und Entwicklungsumgebungen, die sich im Rahmen der jeweiligen Methodologie einsetzen lassen. Desweiteren wird der *praktische Einsatz* der Methodologie betrachtet, etwa die tatsächliche Verwendung innerhalb verschiedener Projekte, die Akzeptanz durch externe Entwicklergruppen, der Einsatz der damit entwickelten Ontologien in realen Systemen, usw.

Eine Zusammenfassung der Ergebnisse der Studie dient im nächsten Abschnitt als Ausgangspunkt, anhand neuerer Arbeiten die aktuelle Evolution von Methodologien zu skizzieren.

Evolution von Methodologien und aktuelle Forschungsschwerpunkte

Die Studie von Fernández-López und Gómez-Pérez (vgl. ebd.), die sich auch in einer erweiterten Fassung bei Gómez-Pérez et al. (Gómez-Pérez et al., 2003, S.148-154) finden lässt, ist eine analytische Untersuchung, die die sieben in Tabelle 1.1 aufgeführten Methodologien nach den vorgestellten Kriterien (s. o.) untersucht und miteinander vergleicht. Wie der Tabelle zu entnehmen ist, stammen alle untersuchten Methodologien aus der Zeit 1995-2001, was sie zu Pionierarbeiten in diesem Bereich zählen lässt. Für eine ausführliche Beschreibung der einzelnen Methodologien wird auf die angegebenen Referenzen verwiesen; hier wird nur das Bild zusammengefasst, das in der Studie über den Stand der damaligen Methodologien gewonnen wurde.

Methodologie	Referenzen
CyC	Lenat and Guha, 1990
Uschold und King	Uschold and King, 1995; Uschold, 1996
Grüninger und Fox	Grüninger and Fox, 1995
KACTUS	Bernaras et al., 1996
Methontology	Fernández-López et al., 1997
Sensus	Swartout et al., 1996
On-To-Knowledge	Staab et al., 2001

Tabelle 1.1: Von Fernández-López und Gómez-Pérez (vgl. Fernández-López and Gómez-Pérez, 2002) untersuchte Methodologien

Die untersuchten Methodologien (s. Tab. 1.1) konzentrieren sich allesamt auf die Entwicklung von Ontologien von Grund auf und weniger auf die Wiederverwendung und Adaption bereits bestehender (*reuse*). Dementsprechend nehmen sie Fokus auf die Beschreibung der Entwicklungsaktivitäten im engeren Sinne (*ontology development oriented activities*), wobei Aspekte wie Management oder Evaluation gar nicht angesprochen werden oder höchstens eine untergeordnete Rolle spielen. Letzteres ist der Fall bei den METHONTOLOGY und On-To-Knowledge-Methodologien, die vom Umfang der beschriebenen Aktivitäten schon als modern angesehen werden können. Die meisten Methodologien in der Studie geben keinen spezifischen Ontologielebenszyklus an; der Beschreibung nach lässt

sich jedoch vom Modell der *evolving prototypes* ausgehen (Gómez-Pérez et al., 2003, S. 153-154). Bezüglich der Strategie zur Identifikation und zum Aufbau der Taxonomie als Kern der Ontologie sind alle drei in Frage kommenden Strategien vertreten, wobei die *middle-out*-Vorgehensweise vorherrscht. Bei der Ontologieentwicklung bleibt das Verhältnis zu potenziellen Anwendungen zunächst offen, da alle drei diesbezüglich definierten Strategien vertreten sind. Zum Zeitpunkt der Studie hatten alle Methodologien einen beschränkten praktischen Einsatz erfahren, da viele nicht von externen Entwicklergruppen verwendet wurden oder sich auf eine einzige Wissensdomäne oder ein einziges Anwendungsszenario beschränken.

Etwa die Hälfte der Methodologien nehmen keinen expliziten Bezug auf ein Tool, das sie bei der Durchführung der verschiedenen Aktivitäten unterstützt. Die CyC-Methodologie beschreibt eine Reihe speziell entwickelter Werkzeuge; im Zentrum der SENSUS-Vorgehensweise steht das Programm Ontosaurus, bei On-To-Knowledge ist es die Entwicklungsumgebung *OntoEdit* mit verschiedenen optionalen Erweiterungen (Plug-Ins). Das ganze Spektrum der Aktivitäten wird jedoch von keinem beschriebenen Programm abgedeckt.

Ein Einblick in neuere Methodologien zeigt klare Tendenzen auf der theoretischen Ebene auf, selbst wenn man die oben erwähnten Analysekriterien nicht systematisch anwendet. Für die folgende, knappe analytische Charakterisierung aktuellerer Methodologien wurden die in Tabelle 1.2 aufgelisteten Arbeiten herangezogen.

Alle dort aufgeführten Arbeiten sprechen einen Ontologie-Lebenszyklus an, der als iterativer, zirkulärer Prozess verstanden wird, dem ein Modell von *evolving prototypes* zugrunde liegt. Charakteristisch ist auch, dass die Einbindung der Ontologie innerhalb einer Anwendung in diesem zirkulären Prozess einen festen Platz hat (HCOME, DILIGENT, Luczak-Rösch und Heese). Dies ist ein deutlicher Unterschied zu den Ergebnissen der Studie von Fernández-López und Gómez-Pérez (vgl. Fernández-López and Gómez-Pérez, 2002), in der noch alle drei Strategien bzgl. der Anwendung vertreten sind. Auf die Rolle der praktischen Anwendung der Ontologie im Entwicklungsprozess wird im nächsten Abschnitt (S. 18 ff) separat eingegangen. Ein fester Bestandteil des iterativen Entwicklungsprozesses ist ebenfalls eine Evaluationsphase, in der die bis dahin entwickelte Ontologie bewertet wird. Hierfür lassen sich jedoch unterschiedliche Kriterien ansetzen, sodass von keiner etablierten Vorgehensweise gesprochen werden kann. Einen Überblick über Evaluationsverfahren bietet der Abschnitt 1.3.4.

Das Hauptaugenmerk der aktuelleren Methodologien hat sich weg von der ursprünglichen Entwicklung von Grund auf durch einen kleinen Kreis von Ontologie-Experten hin zu Evaluation, Maintenance und weiterem Ausbau durch den Kreis der Ontologie-Anwender verlagert. Diese werden letzten Endes als die Experten im jeweiligen Wissensbereich angesehen (DILIGENT, Ontology Maturing). Wichtige Motivationen hierbei sind, den Aufwand der Entwicklung insgesamt auf mehrere Personen zu verteilen, sowie frühzeitig die Akzeptanz der Ontologie zu sichern (Consensus Building). Damit sind auch die betrachteten Entwicklungsszenarien explizit verteilt und oft kollaborativ, im Sinne flacher hierarchischer Strukturen. Das ist ein klarer Gegensatz zur Studie von Fernández-López und Gómez-Pérez (vgl. ebd.), in dem mögliche Strategien diesbezüglich noch nicht betrachtet werden. Die angegebenen Referenzen thematisieren verschiedene Gestaltungsmöglichkeiten

Methodologie	Referenzen
Consensus Building	Karapiperis and Apostolous, 2006
HCOME	Vouros et al., 2007
DILIGENT	Vrandecic et al., 2005; Pinto et al., 2004
Ontology Maturing	Braun et al., 2007
Luczak-Rösch und Heese	Luczak-Rösch and Heese, 2008

Tabelle 1.2: Einige aktuellere Methodologien zum Aufbau von Ontologien

des Entwicklungsprozesses in einem solchen verteilten, kollaborativen Szenario.

Aus einer technologischen Perspektive haben sich seit dem Bericht von Fernández-López und Gómez-Pérez (vgl. ebd.) deutliche Weiterentwicklungen bezogen auf die Unterstützung der mit dem Ontologielebenszyklus verbundenen Aktivitäten ergeben. Dies ist nur zum Teil aus den Arbeiten in Tab. 1.2 ersichtlich. Heuzutage ist eine Reihe unterschiedlicher Ontologie-Editoren verfügbar, mit denen zumindest die Formalisierungsphase bewältigt werden kann. Auch wenn, wie es bei Fernández-López und Gómez-Pérez (vgl. ebd.) noch hieß, kein System würde alle Aktivitäten des Lebenszyklus abdecken, ist die Tendenz zu integrierten Entwicklungsumgebungen, in denen auch neue Techniken zur Unterstützung von *ontology support*-Aktivitäten wie etwa die Alinierung oder das automatische Lernen von Ontologien eingebunden werden, unverkennbar. Nähere Details der technologischen Unterstützung des Ontologieentwicklungsprozesses werden später in Abschnitt 1.3.3 besprochen.

Spezifische Aspekte des Managements eines Ontologieentwicklungsprojektes sind in den angegebenen Referenzen und in der Literatur zum Ontology Engineering generell unterrepräsentiert. Nennenswert in diesem Bereich ist das Kostenmodell für die Entwicklung von Ontologie ONTOCOM (vgl. Simperl and Sure, 2008), in dem zumindest der Frage der Ressourcen nachgegangen wird. Dieses Modell wurde exemplarisch auf die Methodologie DILIGENT angewendet (vgl. Bontas and Tempich, 2005).

Die in Tab. 1.2 angegebenen Methodologien haben einen begrenzten Einsatz erfahren. In der Literatur ist aber zu beobachten, dass Entwicklungsszenarien im unternehmerischen Bereich zunehmend wahrgenommen und damit nicht-akademische Rahmenbedingungen stärker berücksichtigt werden, wie in der Methodologie von Luczak-Rösch und Heese (vgl. Luczak-Rösch and Heese, 2008).

Der Ontologielebenszyklus und die Rolle der Anwendung

Wie im vorherigen Abschnitt angedeutet, gehen aktuelle Methodologien davon aus, dass die Entwicklung einer Ontologie ein zirkulärer, iterativer Prozess ist, in dem sich verschiedene Phasen unterscheiden lassen. Die verschiedenen Vorschläge für die Gestaltung dieses Prozesses unterscheiden sich im Detail, das Grundschema bleibt jedoch stets ähnlich. Stellvertretend für die unterschiedlichen Varianten wird hier ein Ontologielebenszyklus vorgestellt, der sich an die Beschreibung von Buitelaar (vgl. Buitelaar, 2007) anlehnt:

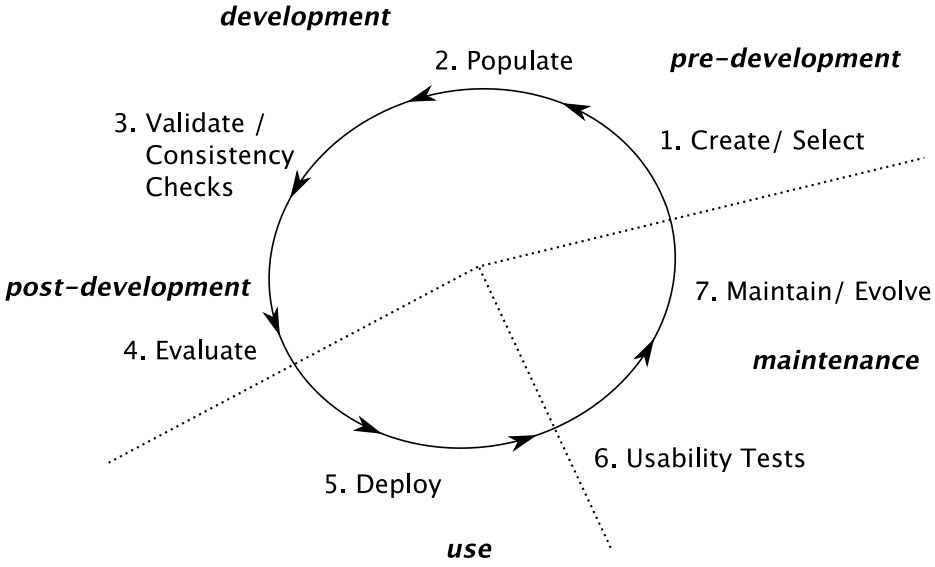


Abbildung 1.3: Generischer Lebenszyklus einer Ontologie. Angelehnt an Buitelaar (vgl. Buitelaar, 2007).

1. *Create/ Select*: Erfassung von Anforderungen, Erstellung einer Kernontologie der wichtigsten Objektklassen.

2. *Populate*: "Bevölkern" der Ontologie, etwa durch Angaben der Instanzen zu den verschiedenen Objektklassen, durch Ontologie-Merging, durch Einbindung bestehender Daten, usw.

3. *Validate/ Consistency Checks*: Überprüfung des formalen Aspekts der Ontologie, z.B. sicherstellen, dass sich für Instanzen keine Konflikte ergeben, dass es keine leeren Klassen gibt, usw.

4. *Evaluate*: Evaluation der Ontologie.

5. *Deploy*: Einbindung der Ontologie in eine konkrete Anwendung.

6. *Usability Tests*: Testen im Rahmen der Anwendung, Berichtigung, Erweiterung bzw. Anpassung an neue Daten.

7. *Maintain/Evolve*: Verfeinerung des Modells nach neu gewonnenen Einsichten.

Fest eingeplant in diesem Zyklus ist immer die Einbindung der Ontologie in eine konkrete Anwendung – das *Deploy* im obigen Ablauf. Die Aktivitäten vor dem *Deploy* korrespondieren in etwa mit den bereits angesprochenen *pre-development* und *development*

Aktivitäten der METHONTOLOGY (vgl. S. 13 ff), wobei hier noch vor der Übergabe der Ontologie ein Evaluationsschritt eingeschaltet wird. Die Schritte danach sind den *post-development* Aktivitäten gleichzusetzen. Der Kreis spiegelt die Annahme wider, dass der Einsatz der Ontologie im Rahmen einer Anwendung ständige Änderungen an der Ontologie motiviert. Der Grund für die Einbindung der Anwendung im Ontologielebenszyklus ist, dass innerhalb der Anwendung eine Übergabe der von den Ontologieentwicklern erarbeiteten Konzeptualisierung an den jeweiligen Nutzerkreis erfolgt. Damit findet eine Überprüfung statt, ob diese Konzeptualisierung geteilt wird. Dementsprechend hat sich der Fokus der aktuellen Methodologien verlagert von der Entwicklung neuer Ontologien durch ein Expertenteam hin zur Anpassung und Weiterentwicklung durch ein Kollektiv von Anwendern. Man kann dabei unterschiedliche Rollen der Anwendung erkennen.

Anwendung als Test: Im oben dargestellten Lebenszyklus könnte man den Einsatz einer Anwendung als einen Übergang zwischen zwei übergreifenden Phasen ansehen: Einer Entwicklungsphase im engeren Sinne und einer Maintenance-Phase, in der die Erkenntnisse aus der Anwendung der Ontologie eingearbeitet werden. In diesem Fall dient die *Anwendung als Testrahmen* der entwickelten Methodologie. Die Gruppen der *ontology engineers* und der Anwender bleiben dabei im Prinzip getrennt: Letztere meldet Probleme an die erste, die sie dann behebt. Die Versionierung der Ontologie ist mit dieser Vorgehensweise noch möglich. Somit spielt die Anwendung die Rolle eines Tests, wie z.B. im Lebenszyklus von Buitelaar (vgl. edb.) oder in der On-To-Knowledge-Methodologie. Oft bleibt die Einbindung der Ontologie in eine praktische Anwendung der einzige durchgeführte Test – die Schritte *Evaluate* und *Usability Tests* verschmelzen zu einem.

Anwendung als Teil der Entwicklung: Ein anderer Ansatz besteht darin, einen Teil der Entwicklung bereits in der Anwendung stattfinden zu lassen. Nachdem ein wesentlicher Part der Ontologie bereitgestellt wird, erhalten die Endnutzer die Möglichkeit, sie innerhalb einer bestimmten Anwendung weiter zu verfeinern oder zu vervollständigen. Hierfür wird die Anwendung in ihrer Nutzen bringenden Funktionalität – bspw. das Finden bestimmter Informationen in einer Dokumentsammlung – erweitert, etwa um einfachere Editierungsmöglichkeiten oder Feedbackmechanismen an die Ontologieentwickler bereitzustellen. Eine wesentliche Motivation dieses Ansatzes liegt darin, den Test, ob die Konzeptualisierung vom Anwenderkreis akzeptiert wird, möglichst früh im Lebenszyklus durchzuführen. Eine weitere Motivation kann die sein, den Entwicklungsaufwand über den größeren Kreis der Anwender zu verteilen. Die DILIGENT-Methodologie vertritt diesen Ansatz; die im NeOn-Projekt (vgl. S. 28) entwickelte, generische Architektur eines ontologiebasierten Informationssystems (vgl. Tran et al., 2007) sieht ihn ebenfalls vor. Eine Folge der Weiterentwicklung durch Anwender ist, dass sie nach dem Modell der *evolving prototypes*[2] stattfinden muss, da jederzeit Änderungen an der Ontologie erlaubt sind. Die Folge dieses Prozesses, nämlich die Evolution der Ontologie, wird manchmal stellvertretend als Bezeichnung für den ganzen Ansatz verwendet. Methoden

[2]Eine technische Definition dieses Begriffes im Kontext des Ontology Engineering findet sich bei Noy und Klein (vgl. Noy and Klein, 2004)

zur Handhabung der Änderungsvorschläge und zur Erhaltung der Konsistenz der Ontologie wurden von De Leenheer und Mens (vgl. De Leenheer and Mens, 2008) erarbeitet.

Anwendung als Katalysator für die Entwicklung: Eine weitere Möglichkeit besteht darin, den Endnutzern einer ontologiebasierten Anwendung die Entwicklung der Ontologie von Grund auf zu überlassen. Meistens wird dieser Ansatz bei Anwendungen verfolgt, in denen die Nutzer selbst den Inhalt kollaborativ bereitstellen. Dazu gehört dann auch die Ontologie, die einen Teil der Funktionalität der Anwendung ermöglicht. Dies ist etwa der Fall bei Semantic Wikis (vgl. Schaffert et al., 2007), eine Erweiterung von sogenannten Wikis, Web-Systemen, mit denen Benutzer Webseiten gemeinschaftlich erstellen und aktualisieren können. Bei einem Semantic Wiki versehen die Autoren, die als Experten des im Wiki erfassten Wissensbereich angesehen werden, die Seiten mit Annotationen. Diese entsprechen dem Vokabular einer Ontologie und können von Benutzern selbst definiert werden, wobei sich teilweise sogar Relationen zwischen den verschiedenen Vokabularelementen definieren lassen. Wichtig ist dabei, dass diese Entwicklung der Ontologie in einem Semantic Wiki ohne den mit einer Ontologiesprache verbundenen technologischen Zwang selbst von nicht versierten Nutzern vorgenommen werden kann. Die so entstandene Ontologie, die intern durchaus mit Hilfe einer Ontologiesprache gespeichert werden kann, bringt der Anwendung zusätzlichen Nutzen, indem beispielsweise Navigation und Strukturierung der Inhalte verbessert werden oder eine semantische Suche ermöglicht wird, sie lässt sich aber auch im Prinzip woanders weiterverwenden. In diesem Ansatz spielen die Endnutzer die Rolle der Ontologieentwickler; *ontology engineers* werden höchstens nachgeschaltet, um das Ergebnis weiter zu formalisieren oder nachträglich zu bearbeiten. Auf kollaborative Ontologieentwicklungsszenarien wird im nächsten Abschnitt näher eingegangen.

Ein weiterer Grund dafür, dass aktuell diskutierte Methodologien allgemein anwendungsorientiert oder -angelehnt sind, könnte die Tatsache sein, dass ontologiebasierte Anwendungen heute deutlich verbreiterter sind als in der Zeit, in der die ersten Methodologien entstanden sind. Dies gilt insbesondere für Anwendungen im Bereich Knowledge Management, die von Unternehmen immer mehr als wichtiger Wettbewerbsfaktor betrachtet werden und um die ein Teil der Softwareindustrie formiert ist.

Dennoch findet faktisch die anwendungsunabhängige Entwicklung von Ontologien statt, nämlich in Bereichen wie Biologie oder Medizin, in denen standardisierte Vokabulare eine lange Tradition und in vielen Fällen normativen Charakter haben. Ein Beispiel hiervon ist die Internationale Klassifikation der Krankheiten (ICD), die für die Verschlüsselung von Diagnosen gesetzlich vorgeschrieben ist. Die Versionierung der Ontologie in diesen Wissensbereichen ist üblich. So ist die aktuelle Version des Functional Catalogue (vgl. MIPS, 2007), eine Ontologie für die Annotation von Proteinfunktionen, die vom Munich Information Center for Protein Sequences (MIPS) veröffentlicht wird, die Version 2.1.

Konsensbildung und Ontologieevolution in kollaborativen Szenarien

Die von Fernández-López und Gómez-Pérez (vgl. Fernández-López and Gómez-Pérez, 2002) berücksichtigten Methodologien thematisieren noch nicht explizit, dass die Entwicklung einer Ontologie in der Regel von mehreren Personen durchgeführt wird. In den aktuelleren Methodologien ist es dagegen üblich, sowohl die Entwicklung der Ontologie vor der Einbindung in die Anwendung als auch die durch die Interaktion mit den Endnutzern ausgelöste Evolution der Ontologie als kollaborative Szenarien zu betrachten. Hier können mehrere Personen gleichzeitig auf die Ontologie zugreifen und sie modifizieren, ohne notwendigerweise vorher Absprache miteinander gehalten zu haben. Solche Szenarien ergeben sich natürlicherweise bei Client-Server-Architekturen, in denen die Anwender räumlich verteilt sind. Manchmal wird dies ausdrücklich angestrebt, um eine Arbeitsteilung zu erreichen und im besten Falle "von der Intelligenz der Masse" zu profitieren.

Dabei stellt sich die Frage, wie ein (idealer) Zustand erreicht werden kann, in dem Konsens über die Ontologie herrscht. Dies wäre ein Zustand, in dem alle Beteiligten eine bestimmte Konzeptualisierung der Domäne teilen, ihre Formalisierung im Rahmen der Ontologie verstehen und sich damit als gültiger Repräsentation der Konzeptualisierung einverstanden erklären, d.h sich zur Ontologie *commiten*. Je nachdem, wie weit man von diesem Zustand entfernt ist, spricht man von einer höheren oder niedrigeren Akzeptanz der Ontologie. In der Praxis zeigt sich, dass das Erreichen von Konsens über die Ontologie oder zumindest einer hohen Akzeptanz wichtiger ist als das Erreichen eines hohen Wahrheitsgehalts oder eines hohen Grades der Formalisierung. So kann etwa im Bereich der Bioinformatik beobachtet werden, dass sich als Standard für die Annotation von Proteinfunktionen eine *light-weight*-Ontologie, nämlich die Function Ontology in GO, etabliert hat, die als azyklischer, gerichteter Graph wenig formalisiert ist, jedoch kollaborativ entstand und daher breit akzeptiert wird. Vergleichsweise bleibt die Verwendung für denselben Zweck von der bereits erwähnten FunCAT eher eingeschränkt und dies trotz der höheren Detailliertheit, der Formalisierung in einer DL-angelehnten Ontologiesprache und der Herausgabe seitens einer wissenschaftlichen Institution.

Wie kann der soziale Verhandlungsprozess, in dem sich Konsens bilden soll, gestaltet werden? Euzenat (vgl. Euzenat, 1995) hat schon 1995 ein Protokoll entworfen, das sich an der von wissenschaftlichen Publikationen eingesetzten *peer-review*-Methode zur Annahme von Artikeln orientiert und diese im Rahmen eines Systems zum kollaborativen Aufbau von Wissensbasen eingesetzt. Ähnlich schlagen Karapiperis und Apostolous (vgl. Karapiperis and Apostolous, 2006) in ihrer Methodologie zum *Consensus Building* vor, die Ontologie in mehreren Durchgängen von den Entwicklern selbst zu bewerten und evaluieren, bis sich alle mit der Ontologie einverstanden erklären.

In anderen Methodologien greifen ausgewiesene *ontology engineers* in die von Endnutzern durchgeführte kollaborative Entwicklung ein. So dürfen in der DILIGENT Methodologie (vgl. Pinto et al., 2004; Vrandecic et al., 2005) die Anwender nach dem Deploy in einer Anwendung die Ontologie nach den eigenen Bedürfnissen editieren. Diese Änderungen werden von einem Kreis von Experten beobachtet, die entscheiden, welche davon in die Ontologie aufgenommen werden. Beim *Ontology Maturing* (vgl. Braun et al., 2007) ist der

Ausgangspunkt ebenfalls eine Anwendung, innerhalb der ein Kreis von Anwendern ein Vokabular und eine einfache Struktur aufstellt und sukzessiv weiterentwickelt. Erst ab einer gewissen "Reife" wird dann dieses Vokabular als Ontologie von einem Kreis von Experten formalisiert.

Eine Reihe von neueren Anwendungen im Internet setzt auf die gemeinschaftliche Beteiligung der Endnutzer, um Inhalte zusammenzutragen und öffentlich zugänglich zu machen. So können Internetnutzer beispielsweise bei del.icio.us (vgl. del.icio.us, 2008) Bookmarks, bei Flickr (vgl. Flickr, 2008) Fotos oder bibliographische Referenzen bei BibSonomy (vgl. BibSonomy, 2008) selbst in einer gemeinsamen, offenen Datenbank anlegen. Die Nutzer versehen die Inhalte selbst mit eigenen Annotationen (*tagging*), um sie auffindbar zu machen und mit anderen Anwendern zu teilen. Die Menge der verwendeten Schlagwörter wird als *Folksonomy* bezeichnet und kann als ontologisches Vokabular aufgefasst werden. Folksonomies stellen jedoch wegen ihres Mangels an Struktur und Formalisierung im Allgemeinen keine Alternativen zu Ontologien dar, vielmehr ist es denkbar, dass sich das *tagging* durch den Einsatz von Ontologien organisieren lässt (vgl. Gruber, 2007). Dennoch gibt es Ansätze, die in Folksonomies den Keim einer Ontologie sehen: Durch die Analyse von Korrelationen zwischen *tags* (vgl. Schmitz et al., 2006) oder die Annotation der *tags* selbst (vgl. Tanasescu and Streibel, 2007) sollen in der Folksonomy Muster und Strukturen erkannt werden, die Relationen einer Ontologie darstellen – die sogenannte emergente Semantik der Folksonomy.

Insgesamt scheinen die kontrollierte Entwicklung durch einen Kreis von Experten einerseits und die wenig formalisierte und spontane Bereitstellung von Wissen seitens der Anwender anderseits komplementäre Aktivitäten zu sein. So merkt Mika (vgl. Mika, 2005) an, dass Folksonomies das Potenzial haben, Neologismen und Änderungen im Wortgebrauch zu erfassen und nur langsam evolvierende linguistische Ontologien wie etwa WordNet zu bereichern und zu vervollständigen. Eine besonders sinnvolle Arbeitsteilung scheint die zu sein, in der Formalisierung, Qualitätssicherung und Standardisierung von einem Kreis von Ontologieexperten übernommen werden, während der Anwenderkreis das Wissen bereitstellt oder die erarbeitete Konzeptualisierung aktuell hält. Dadurch, dass die Anwender weder mit der Hürde einer Ontologiesprache noch mit einem fremddefinierten Vokabular konfrontiert werden, soll für die Akzeptanz der Ontologie gesorgt werden.

Andere Erkenntnisse aus empirischen Studien

Die Analyse und der Vergleich verschiedener Methodologien eröffnen verschiedene Gestaltungsmöglichkeiten für unterschiedliche Prozesse des Ontologieaufbaus und liefern somit Hinweise dafür, was von einem theoretischen Standpunkt aus als gute Praxis anzusehen ist. Dem lassen sich Ergebnisse empirischer Studien gegenüberstellen, die Daten über tatsächlich durchgeführte Ontologieentwicklungsprojekte erheben. Eine der wenigen solcher Studien ist die von Bontas und Tempich (vgl. Bontas and Tempich, 2006), in der Teilnehmer aus 34 unterschiedlichen Ontologieprojekten befragt wurden. Diese Studie verdeutlicht die Abweichung zwischen dem, was von Projektbeteiligten als besonders wichtige oder komplexe Schritte wahrgenommen werden, und der Aufmerksamkeit, die diesen Auf-

gaben in der einschlägigen Literatur zum *Ontology Engineering* gewidmet wird.

Um auf weitere Aspekte hinzuweisen, die in der Praxis besonders zu beachten sind, werden hier die Prozesse wiedergegeben, die Bontas und Tempich (vgl. edb.) als methodologisch wenig unterstützt herausstellen:

- *Analyse der Anwendungsdomäne und Wissensakquisition*: Die Leitlinien, die hierfür von den unterschiedlichen Methodologien vorgeschlagen werden, werden von den Befragten als zu generisch empfunden; insgesamt scheint dieser Schritt kaum operationalisiert zu sein. Insbesondere scheint es keine klare Vorgehensweise zu geben bei der Kombination der unterschiedlichen Techniken und bezüglich des Zeitpunktes, wann Teilergebnisse integriert werden sollen. Dies ist besonders dann der Fall, wenn die Ontologie mehrere Wissensdömanen abdecken soll.

- Die *Ontology Population* wird dadurch erschwert, dass es keine verfügbaren Tools zum Extrahieren ontologischer Strukturen aus bestehenden, in vielen Fällen als semistrukturierten Daten vorliegenden Ressourcen gibt. Ebenfalls wird die fehlende technische Unterstützung bei der Extraktion von Instanzen aus Texten beklagt.

- Eine Evaluation der Ontologie wird in der Regel nicht durchgeführt, da es im Allgemeinen unklar ist, wie und wann diese stattfinden soll. Das Fehlen einer inkrementellen Vorgehensweise zur Evaluation entlang des ganzen Entwicklungsprozesses wird bemängelt. Dies stellt einen klaren Unterschied zu den aktuellen vorgeschlagenen Methodologien dar, die immer einen Evaluationsschritt vorsehen.

- Eine Phase der *ontology maintenance* wird in den meisten Projekten nicht explizit durchgeführt, was mit dem Fehlen einer anerkannten Evaluationsmethode begründet wird: Es ist nicht klar, wann der Übergang von der Entwicklung zur Maintenance stattfinden soll.

Darüber hinaus merken Bontas und Tempich (vgl. ebd.) an, dass den meisten Projektveranwortlichen nicht bewusst ist, dass sich der Prozess des Ontologieaufbaus systematisch organisieren lässt. So folgte keines der befragten Teams einer bestimmten Methodologie. Das wurde mit aus der Tatsache erklärt, dass keine Methodologie alle Aktivitäten abdeckt, auch wenn jede davon zumindest in einer bestimmten Methodologie beschrieben wird. Die Studie bestätigt, dass unterschiedliche Projektumgebungen entsprechende, flexible Zusammenstellungen von Methoden erfordern, im Gegensatz zu den in der Literatur vorgeschlagenen eher rigiden Abläufen – eben eine Methodologie im engeren Sinne des Wortes.

1.3.3 Technologische Unterstützung des Ontologie-Lebenszyklus'

Die Literatur zu Systemen, die unterschiedliche Phasen des Ontologie-Lebenszyklus' unterstützen, ist sehr umfangreich. Sie lässt sich grob in Bereiche unterteilen wie etwa Arbeiten zu Editoren und Inferenzmaschinen, zur automatischen Gewinnung von Ontologien aus

Daten (*ontology learning and population*), zur Alinierung und Integration von Ontologien (*ontology alignment and merging*) sowie zur Wissensakquisition. Es würde den Rahmen dieser Arbeit sprengen, näher auf die unterschiedlichen Bereiche einzugehen.

Um jedoch einen Eindruck über den aktuellen Stand der Technologien zur Unterstützung des Ontologie-Lebenszyklus' zu vermitteln, wird nochmals auf die empirische Studie von Bontas und Tempich (vgl. ebd.) zurückgegriffen. Vor dem Hintergrund der Bewertung der verfügbaren Technologien, die sich aus der in der Studie durchgeführten Befragung für die unterschiedlichen Entwicklungsphasen ergab, wird ein kurzes Bild über den jeweiligen Bereich mit Hinweisen auf neuere Entwicklungen gegeben. Die von Bontas und Tempich (vgl. edb.) betrachteten Phasen der Ontologieentwicklung lehnen sich an die von Fernández-López et al. (vgl. Fernández-López et al., 1997) beschriebenen Aktivitäten an (s. S. 13 ff):

Analyse der Domäne: Die Analyse der anvisierten Domäne ist einer der Bereiche, die sich in der Studie als nicht ausreichend technologisch unterstützt herausstellen. In den befragten Projekten wurden Tools zur Wissensakquisition nicht verwendet oder als nicht nützlich bewertet. Darüber hinaus wurde beklagt, dass verfügbare Tools sich auf die Analyse von Text beschränken, während Extraktion von ontologischen Strukturen aus vorhandenen Ressourcen, insbesondere semistrukturierten Daten, nicht unterstützt wird.

Demgegenüber steht die Aufmerksamkeit, die die Entwicklung von Methoden zur Wissensakquisition in der Forschung verdient. Insbesondere der Bereich des *ontology learning* ist aktuell sehr lebendig, wobei der Fokus doch auf der Gewinnung von Ontologien auf der Grundlage von Textsammlungen liegt; ein aktueller Überblick findet sich bei Cimiano (vgl. Cimiano, 2006). Stellvertretend für eine Vielzahl semiautomatischer Ansätze zur Ontologieextraktion und zur Wissensakquisition sei hier das Text-To-Onto-System erwähnt (vgl. Maedche and Volz, 2001) und die Arbeit von Maedche und Staab (vgl. Maedche and Staab, 2001), die auf die Rolle des maschinellen Lernens zum Aufbau von Ontologien näher eingeht. Ein Beispiel der Verwendung von Internetseiten zur Entwicklung von Ontologien stellen Weber und Buitelaar (vgl. Weber and Buitelaar, 2006) vor.

Bezüglich der Integration strukturierter und semistruktierter Daten in Ontologien sind Systeme beschrieben worden, die von dem Datenbank- bzw. XML-Schema der jeweiligen Ressource ausgehen, um eine Abbildung auf eine bestimmte Ontologie zu berechnen. Beispiele hiervon sind ToMAS oder MapOnto, die bei Euzenat und Shvaiko (Euzenat and Shvaiko, 2007, S. 162 -163) beschrieben sind. Zudem existieren Transformationssprachen wie XSLT (vgl. W3C Consortium, 2006) oder Wrapper-Tools wie Lixto (vgl. Lixto Software GmbH, 2009), mit deren Hilfe sich semistrukturierte Daten zum Zweck der Integration in eine Ontologie aufbereiten lassen.

Konzeptualisierung und Implementierung: Diese Phasen werden nach Bontas und Tempich (vgl. Bontas and Tempich, 2006) gut technologisch unterstützt. Als Grund

hierfür lässt sich die Verfügbarkeit von unterschiedlichen Ontologieeditoren und Inferenzmaschinen angeben, die eine gewisse Reife erlangt haben und zum Teil auch kommerziell vertrieben werden. Unter den Editoren ist das bekannteste Beispiel Protégé, es existieren aber einige andere wie SWOOP, EODM oder unter den kommerziellen etwa OntoStudio, TopBraid Composer und Altova SemanticWorks[3]. Mit Hilfe eines solchen Editors lässt sich eine Ontologie in einer bestimmten Ontologiesprache händisch definieren. Die am meisten unterstützten Sprachen sind RDF/OWL und F-Logic; manche Editoren wie Protégé unterstützen mehr als eine Sprache. Als grundlegender Teil der Entwicklungsinfrastruktur muss meistens zusätzlich zum Editor eine Inferenzmaschine wie Fact++ oder Pellet herangezogen werden, um wesentliche Tests wie etwa Konsistenztests an der im Editor definierten Ontologie durchführen zu können. Die Inferenzmaschine bedingt ebenfalls, welche Ausprägung oder welcher Dialekt der Ontologiesprache während der Entwicklung verwendet werden kann. In Kombination mit Editoren werden zur physischen Speicherung und Abfrage von Ontologien Ontology Repositories eingesetzt. Beispiele hiervon sind Minerva oder Jena.

Ein Grund dafür, dass die Konzeptualisierung, d.h. die Definition eines abstrakten, zunächst nicht formalen Modells als gut unterstützt angesehen wird, ist die Tatsache, dass einige Editoren die Definition einer Ontologie mit Hilfe einer visuellen Sprache erlauben, wie bei Altova SemanticWorks oder EODM mit UML-ähnlichen Diagrammen. Für Protégé sind Erweiterungen erhältlich, die eine Darstellung der Ontologie als Graph erzeugen. Eine solche graphische Editierung der Ontologie ermöglicht zu einem gewissen Grad, von den Details der zugrundeliegenden Ontologiesprache zu abstrahieren. In der Konzeptualisierungsphase lässt sich ebenfalls auf primär nicht für die Ontologieentwicklung gedachte Software zurückgreifen, wie etwa zur Erstellung von Diagrammen oder Mind Maps.

Kritisiert wird an den vorhandenen Editoren, dass sie monolithisch sind, also nur eine bestimmte Aufgabe unterstützen und wenige Schnittstellen zur Kommunikation mit dem Rest der Entwicklungsinfrastruktur anbieten. Darüber hinaus gehen sie sehr stark von dem Bild einzelner Benutzer aus, die getrennt voneinander an einer Ontologie arbeiten, sodass Funktionen vermisst werden, die die kollaborative Entwicklung fördern, wie etwa das Protokollieren verschiedener Vorschläge oder Diskussionen.

Wiederverwendung von Ontologien: Die Studie von Bontas und Tempich (vgl. Bontas and Tempich, 2006) deckt auf, dass die Wiederverwendung von Ontologien und Wissensressourcen prinzipiell in der Praxis nur begrenzt möglich ist. Etwa die Hälfte der Projekte verzichtet darauf und entscheidet sich für den Aufbau einer neuen Ontologie von Anfang an. Ein wesentlicher Grund hierfür ist der Aufwand, der damit verbunden ist, die in Frage kommenden Ressourcen auf die Nützlichkeit für das eigene Projekt hin zu analysieren und zu bewerten. Darüber hinaus werden Tools vermisst,

[3]Literaturverweise zu allen in diesem Abschnitt erwähnten Editoren und Inferenzmaschinen finden sich bei Waterfeld et al. (vgl. Waterfeld et al., 2008), die aktuell verwendete Ontologieentwicklungsumgebungen beschreiben und gegenüberstellen.

die ermöglichen, relevante Teile aus vorhandenen Ontologien oder semistrukturierten Daten zu extrahieren, zwischen Ontologiesprachen zu konvertieren oder leicht in eine andere Ontologie zu integrieren.

Ein Teil der verfügbaren Ontologieeditoren bietet jedoch eine gewisse Unterstützung für diese Aufgaben. So sind für Protégé Erweiterungen wie Prompt (vgl. Noy and Musen, 2000) und Anchor-Prompt (vgl. Noy and Musen, 2001) verfügbar, mit denen sich zwei Ontologien alinieren und ggf. zusammenführen lassen. Im TopBraid Composer können verschiedene Datenformate importiert werden. Die Relevanz der Alinierung und Zusammenführung von Ontologien für die Wiederverwendung von Ressourcen wird in der Forschung zunehmend wahrgenommen, wie das Buch von Euzenat und Shvaiko (Euzenat and Shvaiko, 2007) belegt. In dem Bereich lässt sich auf bekannte Techniken des *schema matching* und *record linkage* zurückgreifen.

Dass mittlerweile Ontologiedatenbanken verfügbar sind, auf die mit entsprechenden Suchmaschinen wie Swoogle (vgl. Li et al., 2004) oder OntoSelect (vgl. Buitelaar et al., 2009) zugegriffen werden kann, dürfte ebenfalls für eine zunehmende Wiederverwendung bereits existierender Ontologien sorgen.

Evaluation: Wie bereits erwähnt, findet in den meisten Ontologieentwicklungsprojekten außer der manuellen Überprüfung keine Evaluation statt. Hierfür gibt es weder standardisierte Vorgehensweisen noch allgemein verfügbare Software. Ein Überblick über akademisch vorgeschlagene Evaluationsmethoden wird in Abschnitt 1.3.4 gegeben.

Ontology Population: Nach Bontas und Tempich (vgl. Bontas and Tempich, 2006) finden sich für die Unterstützung dieser Aufgabe nur begrenzt geeignete Programme. Dies gilt insbesondere für den Ausbau der Ontologie mit Hilfe vorhandener (semi-)strukturierter Daten und Ontologien. Hier kann jedoch gehofft werden, dass ausgereifte Systeme entstehen, die auf Techniken des *ontology alignment/ merging* (vgl. Euzenat and Shvaiko, 2007) aufbauen. In der Forschung konzentriert sich die Aufmerksamkeit auf die "Bevölkerung" von Ontologien mit Daten aus Textcorpora und dem Internet (vgl. Cimiano, 2006).

Ontology Maintenance: Nach den Erkenntnissen von Bontas und Tempich (vgl. Bontas and Tempich, 2006) findet in den meisten Ontologieprojekten aufgrund mangelnder Evaluationskriterien keine bewusste Unterscheidung zwischen Entwicklung und Maintenance der Ontologie statt. In der Literatur finden sich wenige Arbeiten, in denen Systeme oder technische Aspekte dieser Phase beschrieben werden, auch wenn Methodologien wie DILIGENT (vgl. Vrandecic et al., 2005) konkrete Vorschläge hierfür machen. Nennenswert sind Arbeiten zum *change management* von Ontologien in kollaborativen Szenarien (vgl. De Leenheer and Mens, 2008) und das Protégé-PlugIn Prompt-DIFF (vgl. Noy and Musen, 2002), mit dem verschiedene Versionen einer Ontologie miteinander verglichen werden können. Der Ontolingua Server(vgl. Farquhar et al., 1996), ein webbasierter, auf der Sprache KIF aufbauender Editor, unterstützt die Analyse von Änderungen oder die Definition von Modulen, die separat bearbeitet werden können.

Mit dem Text-To-Onto-System (vgl. Maedche and Volz, 2001) wird die semiautomatische Maintenance anhand von Textcorpora demonstriert.

Dokumentation: In den meisten Ontologieeditoren lassen sich Kommentare zu den einzelnen definierten Elementen ablegen. Darüber hinaus ist keine verfügbare Software zur Dokumentation von Ontologien vorhanden, die etwa unterschiedliche Versionen oder kollaborative Aspekte der Ontologieentwicklung berücksichtigt.

Insgesamt betrachtet entstammen die meisten Tools zur Ontologieentwicklung dem akademischen Bereich und fokussieren dementsprechend die Behandlung interessanter Probleme, während einfachere, leicht operationalisierbare Tätigkeiten, die einen wichtigen Teil der praktischen Entwicklungsarbeit ausmachen, kaum technisch unterstützt werden. Im Bereich der Editoren lassen sich jedoch ausgereifte, kommerzielle Systeme finden. Die Feststellung der früheren Studie von Fernández-López und Gómez-Pérez (vgl. Fernández-López and Gómez-Pérez, 2002), dass keine Entwicklungsumgebungen existieren, die das ganze Spektrum der Aktivitäten unterstützen, ist weiterhin gültig. Es wurde jedoch erkannt, dass im *Ontology Engineering* flexible, erweiterbare Entwicklungsumgebungen notwendig sind, die sich an die Bedürfnisse der einzelnen Ontologieprojekte anpassen lassen. Dies zeigt sich etwa an der offenen Architektur mancher Editoren, die sich wie Protégé mit PlugIns zu verschiedenen Aufgaben erweitern lassen, an integrierten Toolkits zur Ontologieentwicklung wie IODT oder an großangelegten wissenschaftlichen Projekten, die wie das NeOn Projekt (vgl. Mehrere Partner, 2008a) die Unterstützung aller Aspekte des Ontologielebenszyklus' als Ziel angeben und generische Software-Architekturen hierfür beschreiben (vgl. Tran et al., 2007) und implementieren (vgl. Mehrere Partner, 2008b). Bereits in vielen Fällen können Editoren, Inferenzmaschinen oder Repositories miteinander interagieren, sodass zumindest ein Kern austauschbarer, kombinierbarer Komponenten vorhanden ist.

1.3.4 Evaluation von Ontologien

Was ist eine gute Ontologie? Einer der ersten Autoren, der eine Antwort auf diese Frage versucht hat, war Gruber (vgl. Gruber, 1995) mit seinen fünf Prinzipien für das Design von Ontologien:

- Klarheit: Die in der Ontologie enthaltene Konzeptualisierung sollte effizient kommuniziert werden, d.h. es sollten Erklärungen und Beispiele angegeben werden, wobei Ambiguitäten zu vermeiden sind; die Begriffe sollen soweit wie möglich mit Hilfe logischer Axiome spezifiziert werden.

- Kohärenz, d.h angegebene Axiome sollen logisch konsistent sein und die Erklärungen und Beispiele durchgehend aufeinander abgestimmt (kohärent).

- Erweiterbarkeit: Eine Ontologie, die sich für unterschiedliche Zwecke leicht erweitern lässt, trifft den richtigen Grad der Abstraktion.

- *mininimal ontological commitment*: Eine Ontologie sollte ausreichend Aussagen machen, die Domäne zu kodieren, zu charakterisieren und somit ungewollte Interpretationen auszuschließen. Die Aussagen über die Domäne sollen aber nicht zu spezifisch sein, sodass die Erweiterung und Wiederverwendung der Ontologie in verschiedenen Anwendungen verhindert wird.

- *minimal encoding bias*: Die Konzeptualisierung der Domäne sollte möglichst unabhängig sein und nicht von den Details der Formalisierung einer spezifischen Ontologiesprache beeinflusst werden.

Diese Kriterien eignen sich jedoch wenig als Grundlage für ein automatisiertes Verfahren zur Evaluation von Ontologien, da sie sich zum Teil nur subjektiv anwenden lassen. Die Notwendigkeit einer automatisierten Evaluation, mit der etwa der Zeitpunkt festgestellt werden kann, an dem von der aufwändigen Entwicklung zur einfacheren Pflege der Ontologie übergegangen werden kann, wurde in der angesprochenen Studie von Bontas und Tempich (vgl. Bontas and Tempich, 2006) aufgezeigt. Darüber hinaus sehen die meisten Methodologien eine Evaluationsphase vor, für die aber keine allgemein anerkannte Methode existiert.

Eine Reihe von spezifischen Methoden zur Ontologieevaluation ist bereits vorgeschlagen worden. Ein Überblick findet sich bei Hartmann et al. (vgl. Hartmann et al., 2005) sowie bei Brank et al. (vgl. Brank et al., 2005). Brank et al. (vgl. edb.) nennen vier verschiedene prinzipielle Ansätze zur Ontologie-Evaluation:

- den Vergleich der Ontologie mit einem *golden standard*, d.h. einer existierenden Referenzontologie,

- die indirekte Evaluation der Ontologie, die über die Evaluation einer bestimmten Anwendung geht, in der die Ontologie verwendet wird,

- die Evaluation auf der Grundlage von Referenzdaten aus dem anvisierten Wissensbereich,

- schließlich die Bewertung der Ontologie durch Personen anhand vordefinierter Kriterien und Anforderungen.

Unabhängig vom gewählten Ansatz kann die Evaluation auf verschiedenen Ebenen stattfinden. Brank et al. (vgl. edb.) geben vier mögliche Evaluationsebenen an: die lexikalische Ebene des von der Ontologie definierten Vokabulars, die relationale Ebene, die taxonomische und weitere Relationen umfasst, die Ebene der Architektur bzw. des Designs, etwa, wie gut die Ontologie erweiterbar ist, sowie die philosophische Ebene, bspw. ob das Kriterium des *minimal ontological commitment* erfüllt ist.

Da EFGT-Netze, wie in Kap. 2 näher erläutert, für den Einsatz in Anwendungen der natürlichen Sprachverarbeitung und des Information Retrieval konzipiert sind, sind im Rahmen dieser Arbeit die Evaluation bezüglich der Anwendung und die auf der Grundlage von Daten aus dem anvisierten Wissensbereich besonders relevant.

Die *anwendungsbezogene Evaluation* einer Ontologie hat die prinzipiellen Nachteile, dass erstens der genaue Beitrag der Ontologie zur Performanz der Anwendung erst einmal ermittelt werden muss und zweitens, dass sich die darin gewonnenen Erkenntnisse nicht unmittelbar auf andere Anwendungen übertragen lassen. Deshalb lässt sich mit diesem Ansatz nicht auf die Güte der Ontologie im Allgemeinen schließen. Außerdem müssten alle Ontologien in derselben Anwendung getestet werden, wenn man ein allgemeines, nachvollziehbares Evaluationskriterium aufstellen wollte. Der Vorteil dieser Vorgehensweise liegt darin, dass man auf Standardmaße zur Evaluation der Anwendung zurückgreifen kann. Um die Maintenance sinnvoll umsetzen zu können ist es außerdem notwendig, die Ontologie im Rahmen der Anwendung fortlaufend zu evaluieren. Ein Tool, das diese Idee aufgreift ist OntoManager (vgl. Stojanovic et al., 2003), das für Anwendungen entworfen wurde, die Ontologien für die semantische Indexierung einer Textsammlung und für die Navigation im Archiv einsetzen. OntoManager analysiert ein Protokoll der Interaktion der Benutzer mit dem jeweiligen System, um Schwachstellen der Ontologie aufzudecken und automatische Änderungen daran vorzuschlagen.

Die *datenbezogene Evaluation* wurde bisher bevorzugt auf Ontologien angewendet, die mittels eines automatischen Verfahrens aus Daten – meistens Textsammlungen – gewonnen wurden (*ontology learning*). Eine datenbezogene Evaluation lässt sich jedoch im Prinzip auch bei manuell erstellten Ontologien einsetzen. Bei dieser Art der Evaluation werden unterschiedliche Ressourcen wie Referenzcorpora, Webseitensammlungen, Lexika oder Datenbanken eingesetzt, die für den von der Ontologie abgedeckten Wissensbereich repräsentativ sind. Hierbei muss zunächst die Frage geklärt werden, was repräsentative Daten bei der anvisierten Domäne genau sind. Die Evaluation erfolgt, indem die "Übereinstimmung" der Ontologie auf einer der erwähnten möglichen Evaluationsebenen mit den Referenzdaten gemessen wird. Da dazu oft Texte herangezogen werden, ist eine entsprechende vorherige Aufbereitung nötig, um den Vergleich etwa auf der lexikalischen oder relationalen Ebene durchführen zu können. Um den Grad der Übereinstimmung zu bestimmen, wird dann je nach Ebene auf unterschiedliche Maße zurückgegriffen, die zum Teil aus anderen Bereichen bereits bekannt sind. So wird etwa bei der Methode EvaLexon (vgl. Spyns and Reinberger, 2005) die Ontologie auf der lexikalischen Ebene mit einer bereinigten Frequenzliste aus einem Referenzcorpus verglichen; die "Übereinstimmung" wird mittels der Maße *precision* und *recall* sowie *accuracy* und *coverage* gemessen, die gängig im Information Retrieval sind. Brewster et al. (vgl. Brewster et al., 2004) setzen dagegen das sogenannte Tennis-Maß (vgl. Stevenson, 2002) ein, das strukturelle Abstände zwischen Termini in einer Taxonomie berücksichtigt. Sie vergleichen damit die Ontologie mit hierarchisch strukturierten Vokabular-Clustern, die aus dem Referenzcorpus extrahiert wurden, sodass in diesem Ansatz neben der lexikalischen auch die relationale Ebene bei der Evaluation miteinbezogen wird.

1.4 Ontologien in der Computerlinguistik und NLP

Bei der Betrachtung unterschiedlicher Arten von Ontologien in Abschnitt 1.3.1 (S. 9 ff) wurde die Gruppe der linguistischen Ontologien zweigeteilt. Eine dieser Untergruppen bilden die Ontologien, die Bezug auf die Primitiva von linguistischen Theorien nehmen: Das Vokabular dieser Art von Ontologien umfasst z.B. unterschiedliche Arten von grammatischen Einheiten, syntaktischen Kategorien, semantischen Rollen, morphologischen Merkmalen, usw. Sie dienen in der Computerlinguistik und im NLP vorrangig dazu, sprachliche Daten zu annotieren. Linguistisch annotierte Daten lassen sich wiederum dafür einsetzen, NLP-Systeme oder Komponenten zu trainieren, etwa ein Part-of-Speech-Tagger auf der Grundlage eines syntaktisch annotierten Korpus'. Annotierte Daten können aber auch das Ergebnis einer Zwischenstufe der automatischen Verarbeitung von Sprache in einem NLP-System darstellen, etwa wenn Texte vom genannten Tagger prozessiert wurden und auf dessen Output hin versucht wird, terminologische Ausdrücke aufgrund bestimmter Folgen syntaktischer Kategorien zu identifizieren.

Wenn man von der automatischen Extraktion und Verarbeitung von Bedeutung aus sprachlichen Daten als einem Ziel vom NLP ausgeht, ist die Verbreitung der zweiten Art linguistischer Ontologien, der "lexikalischen", in diesem Bereich leicht nachvollziehbar. Diese haben eine duale Natur (vgl. Magnini and Speranza, 2002): Sie sind einerseits lexikalische Ressourcen, die sprachliche Ausdrücke einer oder mehrerer Sprachen erfassen und dementsprechend groß sein können. Andererseits wird das darin enthaltene Lexikon auf semantische Konzepte und Relationen abgebildet, dem ontologischen Teil der Ressource. Dadurch wird eine Verbindung zwischen der in der natürlichen Sprache beobachtbaren Ausdrücke mit der damit intendierten Bedeutung (Semantik) hergestellt, was den Nutzen dieser Art von Ontologien im NLP verdeutlicht. Anwendungen von linguistischen, lexikalischen Ontologien sind etwa semantische Indexierung, Informationsextraktion, Disambiguierung. Sie werden ebenfalls in der Maschinellen Übersetzung eingesetzt: Linguistische Ontologien sind als Lexikon stark sprachabhängig, die semantische Schicht wird jedoch als sprachunabhängig oder domänenspezifisch angesehen, sodass sie als eine Art Interlingua zwischen Sprachen vermittelt. Ein früheres Beispiel hiervon ist die Mikrokosmos-Ontologie (vgl. Mahesh and Nirenburg, 1996). Im Projekt EuroWordNet (vgl. Vossen, 1998) wird der semantische Aufbau des englischen WordNet als Struktur zur Alinierung anderer Versionen in unterschiedlichen europäischen Sprachen verwendet.

Linguistische, lexikalische Ontologien sind in der Regel wenig formalisiert. In Einzelfällen wird sogar darüber diskutiert, ob es sich dabei überhaupt um Ontologien handelt. Dies ist insbesondere so für die im NLP-Bereich prominenteste, oft als Ontologie bezeichnete Ressource, das englische WordNet (vgl. Fellbaum, 1998). WordNet ist eine leichtgewichtige Ontologie, die das Vokabular des allgemeinen Englisch anhand von Sinnrelationen wie Synonymie oder Hyponymie strukturiert. Sie wurde ursprünglich jedoch nicht mit Hilfe einer Ontologiesprache explizit formalisiert, auch wenn inzwischen unterschiedliche Alternativen vorgeschlagen wurden (vgl. Huang and Zhou, 2007; W3C, 2006; Ciorăscu et al., 2003). Die mangelnde Formalisierung lexikalischer Ontologien erklärt sich hauptsächlich aus der Größe von solchen Ressourcen, die leicht mehrere tausend lexikalische Einheiten

umfassen können. Weitere inhärente Gründe für diese notgedrungen geringe oder bestenfalls nur leichtgewichtige Formalisierung können anhand der Kriterien von Hepp et al. (s. S. 9) zur Charakterisierung von Ontologien aufgedeckt werden: Neben der großen Anzahl der konzeptuellen Elemente sorgen die hohe Dynamik natürlicher Sprache, die große Nutzergemeinschaft und der hohe Grad der Subjektivität dafür, dass das Aufstellen eines abstrakten Modells sehr schwierig ist und nur eine kleine Durchschnittsgröße der Spezifikation eines jeden konzeptuellen Elements zulässt. Bei lexikalischen Ontologien ist dementsprechend der Aufwand für Pflege und Erhalt der Aktualität hoch.

Viele der domänenspezifischen Ontologien (vgl. S. 10) lassen sich als lexikalische Ontologie auffassen, die die spezifische Terminologie eines bestimmten Wissensbereichs erfassen. Diese Ontologien weisen meistens einen höheren Grad der Formalisierung und ein komplexeres Modell bzw. eine komplexere Struktur auf. Aus der Sicht deren Verwendung als NLP-Ressource ergeben sich in manchen Fällen Probleme aus der Tatsache, dass das darin enthaltene Vokabular – die spezifische Terminologie – nicht genau die tatsächliche Verwendung in der natürlichen Sprache widerspiegelt. Der Grund dafür ist das primäre Interesse der Ontologieentwickler für die Modellierung des Bereichs und nicht für die Erfassung sprachlicher Ausdrücke; letztere stellt ja eine zusätzliche Aufgabe dar, da der sprachliche Verweis auf ein semantisches Konzept stark variieren kann. Darüber hinaus wird oft nicht genügend zwischen der (den) verschiedenen sprachlichen Bezeichnung(en) eines Konzepts und dessen Bezeichner oder Identifikator in der Ontologie unterschieden. Ein Modell, um diese Unterscheidung explizit zu machen, ist LexOnto (vgl. Cimiano et al., 2007), das bspw. bei der Abbildung bestehender linguistisch orientierter, terminologischer Lexika in entsprechenden Domänenontologien eingesetzt werden kann. LingInfo (vgl. Buitelaar et al., 2006) ist ein weiteres Modell zur Integration linguistischer Information in Ontologien.

Eine Motivation für den Aufbau von Ontologien ist die Wiederverwendung im Rahmen unterschiedlicher Anwendungen. In der Praxis sieht es so aus, dass eine Wiederverwendung von computerlinguistischen Ressourcen nicht uneingeschränkt möglich ist. Das hat mit dem enormen Aufwand bei der Erfassung lexikalischer Ausdrücke zu tun, sodass linguistische Ontologien auch immer der natürlichen Sprache nachhinken. Dies ergibt sich auch aus der Dynamik, die durch neue Wortschöpfungen und Annahme von Neologismen der Sprache innewohnt und die einer linguistischen Ontologie immer eine *open world*-Annahme aufzwingt (vgl. S. 44). Oft besteht daher bei der Aufbereitung einer NLP-Ressource für eine spezifische Anwendung eine zentrale Aufgabe darin, auf bestehende, sich ergänzende Ressourcen zurückzugreifen und zu integrieren. Diesem Prozess kommen Techniken des *ontology merging* und des Abgleichs linguistischer Daten zugute.

Wie bereits in Abschnitt 1.3.3 erwähnt, besteht eine weitere Beziehung zwischen automatischer Sprachverarbeitung und Ontologien darin, dass die Methoden, die die automatische Gewinnung oder Erweiterung von Ontologien mit Hilfe textueller Daten anstreben, oft NLP-Techniken einsetzen.

1.5 Schlussfolgerungen und Ausblick

Nach einem kurzen Überblick über Ontologien wurde in diesem Kapitel vor allem auf Methodologien zu deren Entwicklung sowie andere Aspekte des Ontology Engineering, wie Evaluation oder einsetzbare technische Systeme, näher eingegangen. Dabei wurden sowohl aktuelle wissenschaftliche Fragestellungen als auch Erkenntnisse aus der Praxis berücksichtigt. Ein möglichst realistisches Bild der relevanten Aspekte, die zur Verbesserung des Lebenszyklus' einer Ontologie beitragen können, wurde somit umrissen.

Als Resümee lässt sich ziehen, dass es keine feste Methodologie gibt, die in jedem Fall erfolgversprechend ist. Vielmehr sind die verschiedenen Phasen, in die sich der Prozess der Entwicklung einer Ontologie einteilen lässt, an die Anforderungen des jeweiligen Projekts anzupassen. Bei der Betrachtung verschiedener Arten von Ontologien wurden Kriterien diskutiert, mit denen sich ein spezifisches Ontologieprojekt charakterisieren lässt. Unter den unterschiedlichen methodologischen Aspekten, die bei der Definition von Maßnahmen zur Verbesserung eines spezifischen Entwicklungsszenarios zu berücksichtigen sind, lassen sich unter anderem hervorheben:

- Die Relevanz, die die spezifische Unterstützung verteilter und kollaborativer Szenarien für die Entwicklung von Ontologien hat.
- In Ermangelung allgemein einsetzbarer Evaluationsmethoden kann dennoch mit Hilfe einer anwendungsnahen Entwicklung der Entwicklungszustand der Ontologie abgeschätzt und somit eine Steuerung des Entwicklungsprozesses ermöglicht werden.
- In vielen Fällen ist es zielführender, weniger komplexe, dafür praxisnähere Aufgaben zu operationalisieren, so z.B. die Vervollständigung von Ontologien mit Hilfe semistruktierter Daten, anstatt den Fokus auf zwar viel beachtete, jedoch noch nicht kontrollierbare Techniken, wie etwa die Extraktion von Ontologien aus Textsammlungen, zu richten.

Darüber hinaus wurden spezifische Merkmale von Ontologien beschrieben, die im Bereich des NLP als lexikalische Ressourcen eingesetzt werden, wie deren notwendig leichtgewichtige Formalisierung oder der hohe Aufwand, der betrieben werden muss, um sie aktuell zu halten. Hierfür sind spezifische Methoden zu entwickeln.

Aus der Betrachtung der unterschiedlichen Methodologien lässt sich der Schluss ziehen, dass es keine klar definierte Lösung gibt, mit der sich der vielzitierte Engpass der Ontologieentwicklung, der *ontology bottleneck*, überwinden lässt und die Umsetzung der *Semantic Web*-Vision im vollen Umfang ermöglicht. Die Entwicklung und Pflege von Ontologien und Wissensressourcen im allgemeinen ist vergleichbar mit der Entwicklung und Wiederverwendung von Software-Komponenten: Neuentwicklungen und Anpassungen werden bis auf weiteres nötig bleiben. Ähnlich hat sich der Traum der Künstlichen Intelligenz von allgemein verfügbaren, wiederverwendbaren Komponenten, die sich zu immer größeren Systemen zusammenfügen lassen, bis jetzt nicht im ursprünglich angedachten Ausmaß bewahrheitet.

Im nächsten Kapitel wird auf EFGT-Netze eingegangen, eine bestimmte Art leichtgewichtiger, lexikalischer Ontologien mit einem eigenen Formalismus. Nachdem die spezifische Vorgehensweise und vorhandene Infrastruktur zur Entwicklung von EFGT-Netzen darin beschrieben wird, werden in den darauffolgenden Kapiteln unterschiedliche Maßnahmen ausgearbeitet, die auf eine Verbesserung der Entwicklungsumgebung abzielen.

Kapitel 2

Ontology Engineering für EFGT-Netze

Das allgemeine Ziel der vorliegenden Arbeit ist einen Beitrag zu leisten, um den Aufbau und die Pflege von EFGT-Netzen effizienter zu gestalten. Dieses allgemeine Ziel hat zwei Aspekte, nämlich einen methodologischen und einen technischen. Aus der methodologischen Perspektive geht es darum, besonders aufwändige Schritte im Lebenszyklus eines EFGT-Netzes zu identifizieren und Vorschläge zu machen, die die mit diesen Schritten verbundene Arbeit erleichtern. Aus der technischen Perspektive ist das Ziel, die Anforderungen, die methodologisch entstehen, durch Konzeption und Realisierung verschiedener technischer Maßnahmen zu unterstützen.

In diesem Kapitel werden EFGT-Netze zunächst als Formalismus sowie als real existierende Wissensressourcen vorgestellt. Der zweite Teil des Kapitels beschreibt am Beispiel einer realisierten Wissensressource, des CoGE-Netzes, die konkrete Vorgehensweise, die beim Aufbau von EFGT-Netzen verfolgt wurde. Diese Betrachtung vom Lebenszyklus eines EFGT-Netzes dient als Grundlage, technische Ziele zu identifizieren, die im weiteren Verlauf der Arbeit umgesetzt werden und den Lebenszyklus von EFGT-Netzen unterstützen.

Alle Arbeiten rund um den EFGT-Formalismus und realisierte EFGT-Netze wurden am Centrum für Information- und Sprachverarbeitung der LMU München durchgeführt. Grundlegende Literaturreferenzen zu diesem Kapitel sind Schulz and Weigel, 2003; Brunner et al., 2006; Weigel et al., 2006; Brunner, 2008.

2.1 Überblick über EFGT-Netze

2.1.1 Motivation

Die ursprüngliche Motivation für die Entwicklung von EFGT-Netzen ist eine grundlegende computerlinguistische Einsicht: Die zentrale Rolle von *named entities* für die Einordung der Thematik und die Bestimmung des Inhalts eines Dokuments. *Named entities* oder auch benannte Entitäten wie etwa *Mario Vargas Llosa*, *Springer-Verlag*, *Schwabmünchen* oder

Handbuch der Ontologien sind Ausdrücke der natürlichen Sprache, die den allgemeinen Wortschatz einer Sprache erweitern und dazu benutzt werden, auf einen Begriff zu verweisen, das als Individuum spezieller Natur (Mensch, Ort, Buch, Organisation, usw.) aufgefasst wird. Wichtig dabei ist, das es sich bei dem Ausdruck nicht um die allgemeinsprachliche Beschreibung der Entität handelt. So ist der Ausdruck *die von Henry Ford 1903 gegründete Firma* also keine benannte Entität, *Ford* oder *Ford Motor Company* jedoch schon, auch wenn sie alle dieselbe Entität bezeichnen. Benannte Entitäten sind typischerweise Namen wie Eigennamen und auch Namen, mit denen auf Objekte der Natur verwiesen wird, wie biologische Spezies und Substanzen.

Benannte Entitäten kommen in Texten häufig vor und bringen einen reichhaltigen semantischen Hintergrund in den Text ein, da der Leser sie mit einer bestimmten Thematik assoziert und in Beziehung zu anderen benannten Entitäten setzt, d.h. etwas über sie weiß. Diese Hintergrundinformation ist eine Voraussetzung für die thematische Einordnung und das Verständnis des Textes, sodass die in einem Text vorkommenden benannten Entitäten eine Art semantischen Fingerabdrucks des Inhalts bilden.

Um sich diese Einsicht bei der maschinellen Textanalyse und der computerlinguistischen Informationsverarbeitung zunutze zu machen, muss die semantische Information, die mit benannten Entitäten assoziert wird, explizit und maschinell verarbeitbar gemacht werden. Das ist das Ziel eines EFGT-Netzes, wobei der Fokus auf bestimmte Arten von semantischen Informationen gelegt wird. Ein EFGT-Netz ist eine Wissensressource, die einen temporal, geographisch und thematisch strukturieren Raum bietet, in denen benannte Entitäten eingeordnet und dadurch explizit in Beziehung zu anderen Entitäten gesetzt werden können.

Dieser Raum bildet ein logisches Modell von prominenten Entitäten der Welt und zugleich einen Navigationsraum, in der Entitäten in einer thematisch-geographisch-temporalen Hierarchie eingeordnet sind. Die Knoten in dieser Hierarchie, sogenannte *Konzepte*, stellen die benannten Entitäten sowie weitere temporale, geographische und thematische Begriffe dar, die damit verbunden sind und die Hierarchie gliedern. Pfade in dieser Hierarchie kann man mit Assoziationsketten identifizieren, die, beginnend bei einem Konzept zu weiteren "verwandten" führen. Zwei über einen solchen Pfad verbundene Konzepte werden dadurch im Netz explizit in Beziehung gesetzt. Charakteristisch ist hier, dass man zu einem Konzept über unterschiedliche Ketten gelangen kann.

Das logische Modell eines EFGT-Netzes wird durch eine linguistische Ebene ergänzt, die unterschiedliche Namen und Ausdrücke erfasst, mit denen in der natürlichen Sprache auf einzelne Konzepte verwiesen wird. Sie dient dazu, durch Erkennung dieser Ausdrücke das Netz in zu verarbeitenden Texten zu verankern. Auf diesen beiden Ebenen, der logischen und der linguistischen, zeigt sich der hybride Charakter des EFGT-Netzes, das sich als linguistische Ontologie auffassen lässt.

In den folgenden Abschnitten wird der Formalismus, der zum Aufbau von EFGT-Netzen verwendet wird, kurz vorgestellt sowie die Architektur und technische Infrastruktur, die den physischen Aufbau des Netzes ermöglichen. Anschließend werden eine spezielle, als EFGT-Netz realisierte Wissensressource, das sogenannte CoGE-Netz, und intendierte Anwendungen beschrieben.

Die Vorgehensweise oder das *Ontology Engineering* von EFGT-Netzen ist das Thema des zweiten Teils des Kapitels.

2.1.2 Der EFGT-Formalismus

Die Designkriterien, die bei der Entwicklung des EFGT-Formalismus' verfolgt wurden, sind bei Schulz und Weigel (vgl. Schulz and Weigel, 2003) ausführlich beschrieben. Eine Grundmotivation war dabei der Entwurf einer *Systematik* – ähnlich den Klassifikationssystematiken aus dem Bibliothekswesen wie etwa Universal Decimal Classification (vgl. UDC Consortium, 2008), mit der Einträge einer Wissensressource mittels der Definition von *Identifikatoren* in ein abstraktes Ordnungsschema positioniert werden können. Im Fall eines EFGT-Netzes entspricht das Ordnungsschema einem gerichteten, azyklischen Graphen[1]. Die Knoten dieses Graphen stellen dabei die Einträge der Wissensressource dar, die Kanten des Graphen entsprechen Relationen zwischen den jeweiligen Knoten bzw. Einträgen.

Eine weiteres Designkriterium war, dass die Systematik typisiert sein sollte, d.h. sie sollte Typen zur Verfügung stellen, die die grundlegenden Klassen von Einträgen zusammenfassen. Der Name EFGT-Formalismus verweist auf die grundlegenden Klassen von Einträgen, die im Formalismus unterschieden werden. Im Einzelnen sind das entitätsspezifische Einträge (E), thematische Einträge oder auch *thematic Fields* (F), geographische (G) und temporale (T) Einträge. Neben der Unterscheidung nach dem grundlegenden Charakter des Eintrags wird zusätzlich differenziert, ob das Konzpet sich als Menge von Entitäten interpretieren lässt (z.B. *Gewässer*) oder als einzelne Entität (z.B. *der Lech*). Dadurch ergeben sich folgende Typen für die verschiedenen Konzepte:

`E,e:` Der Typ `E` bezeichnet eine Menge von *E*ntitäten – etwa *Komponisten* – während Typ `e` für einzelne Entitäten verwendet wird, z.B. *J. S. Bach*.

`F:` Typ `F` wird für Einträge verwendet, die explizit ein Themengebiet (*thematic F*ield) bezeichnen, etwa *Quantenphysik*. Da jedes Themengebiet gleichzeitig als eine Menge von Themengebieten aufgefasst werden kann, ist der Typ `f` nicht im Formalismus definiert.

`G,g:` Typ `G` wird für Konzepte verwendet, die *g*eographische Sammelbegriffe darstellen und keinen Bezug auf eine konkrete geographische Position haben, wie etwa *Gebirge* oder *Wüsten*. Typ `g` wird für individuelle geographische, auf Karten vorkommenden Gebiete verwendet, z.B. *die Alpen* oder *Sahara*.

`T,t:` Mengen *t*emporaler Zeiträume wie *Erdzeitalter* oder *Kunstepochen* erhalten im Formalismus den Typ `T`, während `t` ein individuelles temporales Intervall wie z.B. das *Kambrium* oder *der 11. September 2001* kennzeichnet.

Die Identifikatoren von Einträgen im EFGT-Formalismus, *ID-Strings* genannt, enthalten zum einen einen der beschriebenen Typen für den jeweiligen Eintrag. Zum anderen

[1] Im Folgenden *DAG* nach der englischen Bezeichnung: *directed acyclic graph*

kodiert der ID-String die Beziehungen zu anderen Einträgen im EFGT-Netz. In diesem Sinne sind ID-Strings beschreibend und spiegeln den temporalen, geographischen und thematischen Gehalt des Konzepts wider. Mit *Kodierung* wird in der vorliegenden Arbeit das "Beschreiben" eines Eintrags durch Angabe eines ID-Strings bezeichnet. Durch die Kodierung eines Eintrags werden implizit die Vorfahrensbeziehungen eines Konzeptes zu anderen in der Wissensressource spezifiziert und dadurch das Konzept im EFGT-Graphen positioniert.

Das Spezifizieren eines ID-Strings erfolgt entsprechend einer wohldefinierten Grammatik, die angibt, welche syntaktischen Operatoren zur Bildung neuer ID-Strings zur Verfügung stehen. Im Rahmen der Geschichte des EFGT-Formalismus' hat diese Grammatik verschiedene Varianten gekannt (vgl. Schulz and Weigel, 2003; Brunner et al., 2006; Weigel et al., 2006). Hier wird Bezug auf die Variante genommen, die beim Aufbau des CoGE-Netzes verwendet wurde und in Abbildung 2.1 dargestellt ist.

$$
\begin{array}{lll}
\mathbb{ID}\mathbb{S}\text{tring} & ::= & \texttt{(}\,\texttt{)} \mid \texttt{(}\,\mathbb{T}\text{yp}\;\mathbb{ID}\mathbb{S}\text{tring}\;\texttt{.}\;\mathbb{I}\text{ndex}\,\texttt{)} \mid \texttt{(}\,\mathbb{ID}\mathbb{S}\text{tring}(\,\texttt{\&}\;\mathbb{ID}\mathbb{S}\text{tring})^{+}\,\texttt{)} \\
\mathbb{T}\text{yp} & ::= & \texttt{e} \mid \texttt{E} \mid \texttt{F} \mid \texttt{g} \mid \texttt{G} \mid \texttt{t} \mid \texttt{T} \\
\mathbb{I}\text{ndex} & ::= & \mathbb{Z}\text{iffer}\;(\,\texttt{0} \mid \mathbb{Z}\text{iffer}\,)^{*} \\
\mathbb{Z}\text{iffer} & ::= & (\,\texttt{1} \mid \ldots \mid \texttt{9}\,)
\end{array}
$$

Abbildung 2.1: Syntax der ID-String-Identifikatoren

Alle ID-Strings bauen auf dem leeren Identifikator `( )` auf. Dieser Identifikator ist immer definiert und bezeichnet den (einzigen) Wurzelknoten des EFGT-Netzes, den sogenannten *Topnode*, der das allgemeinste, alle anderen Konzepte umfassende Thema des Netzes darstellt. Ausgehend vom leeren Identifikator stehen zwei Operatoren, nämlich die *lokale Einführung* und die `&`-*Summe*, zur Verfügung, mit Hilfe derer neue Identifikatoren erzeugt werden können.

Lokale Einführung: Die *lokale Einführung* entspricht der zweiten Alternative der Regel für $\mathbb{ID}\mathbb{S}$tring und erfolgt, in dem ein ID-String ϕ mit einem Typ τ aus $\mathbb{T}$yp und einem Index n (einer natürlichen Zahl) zum neuen ID-String `(`$\tau\phi.n$`)` ergänzt wird. Die Semantik der lokalen Einführung ist, dass das Konzept, dem der neue ID-String `(`$\tau\phi.n$`)` zugewiesen wird, definiert wird als das n-te Unterthema oder Verfeinerung vom Typ τ des Konzeptes, das mit ϕ korrespondiert. Der Index n dient also lediglich dazu, mehrere Unterthemen eines Konzeptes vom selben Typ voneinander zu unterscheiden. Beispielsweise könnte man den Begriff *Städte der Welt* als geographischen Sammelbergiff direkt unter den Topnode mittels des Identifikators `(G().1)` in das Netz einführen und analog *Flüsse* mit dem ID-String `(G().2)`. *Oslo* lässt sich dann so unter dem Konzept *Städte der Welt* mit dem ID-String `(g(G().1).1)` lokal einführen, *Stockholm* als `(g(G().1).2)`. *Probleme der Städte* kann ebenfalls als Unterthema von *Städte der Welt* der Identifikator `(F(G().1).1)` zugewiesen werden. Ein Typenwechsel innerhalb einer Reihe von verschachtelten lokalen Einführungen ist erlaubt. Einen Eindruck über die Möglichkeiten der Beziehung zwischen Basis- und Unterthema bei unterschiedlichen Typenkombinationen vermittelt Brunner (Brunner, 2008, S. 34).

`&`-*Summe*: Die `&`-*Summe*, die dritte alternative Regel für $\mathbb{ID}$String, besteht darin, eine beliebige Zahl von ID-Strings mittels des Operators `&` – gelesen: *und* – zu einem neuen Identifikator `(` ϕ_1 `&` ϕ_2 `&` ... `&` ϕ_n `)` kommutativ zu verbinden. Die ϕ_i der so gebildeten Summe werden in der Folge *Summanden* oder *Komponenten* genannt. Semantisch betrachtet ist die Summe `(` ϕ_1 `&` ϕ_2 `&` ... `&` ϕ_n `)` als ein Konzept aufzufassen, das die Schnittmenge der Konzepte darstellt, die mit den Summanden ϕ_1 bis ϕ_n korrespondieren. In der Topologie des Netzes wird dabei jede Komponente zu einem übergeordneten Konzept des neuen. Beispielsweise könnte man das Konzept *Personen der Politik* als Summe oder "Verundung" der Konzepte *Personen* (z.B. `(E().1)`) und *Politik* (z.B. `(F().1)`) auffassen und ihm den ID-String `((E().1)&(F().1))` geben. Formal gesehen haben `&`-Summen, deren Komponenten desselben Typs sind, denselben Typ wie die Komponenten. In den anderen Fällen entscheiden Regeln über den Typ des Identifikators. So ist *Personen der Politik* vom Typ `E`, weil dieser Typ "stärker" ist als `F`. `F` ist "stärker" als die restlichen Typen: Die Summe von *USA* (vom Typ `g`) und *Wirtschaftspolitik* (vom Typ `F`) ist wiederum vom Typ `F`. `F` ist ebenfalls der Default-Typ bei anderen Typkombinationen.

Um die Notation der ID-Strings zu vereinfachen und die Lesbarkeit zu erhöhen, wurden Schreibkonventionen eingeführt. Beispielsweise kann in einem ID-String der Identifikator für den Topnode weggelassen werden, sodass z.B. `(G().1)` auch `(G.1)` geschrieben werden kann. Ebenfalls können sukzessive lokale Einführungen desselben Typs abgekürzt werden: `(E(((F(F(F().1).3).2).1)` wird vereinfacht `(E(F.1.3.2).1)` geschrieben.

Aus einer Menge von ID-Strings, die nach den vorangehenden Regeln aufgebaut wurden, lässt sich die Struktur des EFGT-Netzes berechnen. Aus der formalen Semantik der ID-Strings (vgl. Brunner et al., 2006) ergibt sich die Tatsache, dass das Netz mathematisch betrachtet einem DAG mit einem einzigen Wurzelknoten, dem Topnode, entspricht. Die Knoten des DAG korrespondieren mit den Einträgen/ Konzepten der Wissensressource; die Kanten im DAG, die jeweils zwei Knoten verbinden, stellen die Elternteil-Kind-Relation dar, wobei das Elternteil das "allgemeinere" Konzept mit dem kürzeren ID-String des Paars darstellt. Die Kanten sind gerichtet und zeigen vom Kind auf das Elternteil. Desweiteren ist die Vorfahre-Nachfahre-Relation als der transitive Abschluss der Elternteil-Kind-Relation definiert. Demnach stellt der Topnode das allgemeinste Konzept oder Vorfahre aller Konzepte des EFGT-Netzes dar. Die Berechnung der beiden Relationen erfolgt mit einem EFGT-Kalkül, das von Brunner et al. (vgl. Brunner et al., 2006) und Brunner (Brunner, 2008, S. 36 ff) ausführlich beschrieben wird. Hierfür müssen die ID-Strings wegen der Kommutativität der `&`-Summe und der Notationskonventionen zunächst in eine Normalform gebracht werden (Brunner, 2008, S. 32 ff). Ein Merkmal des EFGT-Kalküls ist, dass sowohl die globale Berechnung der Netzstruktur für eine gegebene Menge von ID-Strings als auch das Hinzufügen einzelner Knoten zu einem bestehenden Netz durch die Definition neuer Identifikatoren sehr performant implementiert werden kann, so dass große[2] EFGT-Netze aufgespannt werden können.

ID-Strings dienen dazu, Einträge der EFGT-Wissensressource im abstrakten Raum des EFGT-Netzes zu positionieren. Aber wie sehen diese Einträge aus? Neben dem zugehörigen

[2]Bisher liegen Erfahrungen mit EFGT-Netzen mit 10^5 Knoten vor.

ID-String wird ein Konzept in der Wissensressource mit einer *linguistischen Repräsentation* und zusätzlichen *optionalen Attributen* gespeichert. Die linguistische Repräsentation umfasst einen natursprachlichen Namen für das Konzept sowie eine Reihe von Ausdrücken, die in der natürlichen Sprache auf das durch den Eintrag repräsentierte Konzept verweisen. Jede Angabe der linguistischen Repräsentation hat drei Eigenschaften:

Annotationsrolle: Jeder Ausdruck hat eine bestimmte Beziehung zu dem Konzept: Er kann der Name des Konzepts sein, oder ein synonym verwendeter Ausdruck, ein Kommentar zum Konzept, usw. Diese Beziehung wird mit der Annotationsrolle spezifiziert.

Sprache: Hierbei handelt es sich um die natürliche Sprache, der der Ausdruck angehört.

Linguistischer Typ: Der linguistische Typ gibt die morphosyntaktische Wortart des Ausdrucks an, wie etwa *Nominalphrase*, *Nomen*, *Eigennamen*, *Adjektiv*, *Text* (für Kommentare), usw.

Optionale Attribute werden verwendet, um zusätzliche Informationen über das Konezpt zu speichern. Jedes optionale Attribut hat neben dem Wert zwei Eigenschaften:

Annotationsrolle: Die Annotationsrolle von optionalen Attributen können sehr unterschiedlich sein: *Biographie-Link*, *Einwohnerzahl*, *Geburtsjahr*, *Homepage*, usw.

Datentyp: Beim Wert des Attributs kann es sich um einen *String*, eine *natürliche Zahl*, eine *URL*, usw. handeln.

Ein Beispiel für einen Eintrag ist in Abb. 2.3 dargestellt.

Als Schlussbild für die Beschreibung des EFGT-Formalismus' sei das EFGT-Netz als aufgespanntes Fraktal betrachtet (Brunner, 2008, S. 127-128):

Im Netz kann man die einem bestimmten Eintrag unmittelbar untergeordneten Knoten nach den im Formalismus definierten Grundtypen in Mengen zusammenfassen. D.h. unter jedem Konzept im Netz können maximal sieben verschiedene Arten von Einträgen vorhanden sein. Wählt man wiederum einen bestimmten Eintrag aus einer dieser Mengen aus, so kann man unter diesen potenziell dieselbe Struktur vorfinden, in der der Eintrag selbst enthalten ist. Diese sich beliebig oft fortsetzende Wiederholung einer Struktur innerhalb eines Elements der gleichen Struktur kann man mit einem Fraktal identifizieren.

2.1.3 Technik

Die Implementierung des EFGT-Formalismus' und der Infrastruktur zur Speicherung und Entwicklung von EFGT-Netzen beschreibt Brunner (vgl. Brunner, 2008) im Detail. Zum Verständnis sei hier die Architektur des EFGT-Netz-Systems kurz wiedergegeben. Es handelt sich hierbei um eine verteilte, sogenannte *multi-tier*-Architektur, in der folgende Schichten realisiert sind:

***Datenbank-Server* und *Calculus-Server*:** Die beiden Module *Datenbank-Server* und *Calculus-Server* bilden das Herzstück des EFGT-Netz-Systems. Alle mit einem EFGT-Netz verbundenen Daten (Einträge, ID-Strings, abgeleitete Relationen des EFGT-Netzes, usw.) werden persistent im Datenbank-Server abgespeichert. Der Datenbank-Server verwaltet ebenfalls die Berechtigungen für den Zugriff auf das jeweilige EFGT-Netz. Der Calculus-Server ist eine algorithmische Komponente, die alle Operationen betreffend der Struktur des EFGT-Netzes übernimmt, wie die Berechnung der globalen Struktur und das Löschen und Hinzufügen neuer Einträge. Der *Calculus-Server* interagiert mit dem *Datenbank-Server* über eine native Schnittstelle des letzteren.

Das *EFGT-Netz-Interface*: Diese Komponente stellt alle häufig benötigten Vorgänge als Operationen bereit, mit denen auf die vorangehende Schicht zugegriffen und somit mit der Wissensressource interagiert werden kann. Beispielsweise kann über diese Schnittstelle der Administrator des EFGT-Netz-Systems in einer UNIX-Shell Wartungsarbeiten durchführen. Für den Zugriff externer Programme steht diese Schnittstelle sowohl in einer Skriptsprache (Perl) als auch als XML-Dialekt zur Verfügung – das sogenannte *XNet-Interface*.

Der *Web- und der Socket-Server*: Die Wissensressource ist über diese Module, die auf dem EFGT-Netz-Interface aufbaut, über das Internet erreichbar. Der Socket-Server bietet fremden Applikationen eine Möglichkeit, per SOAP und XML das *EFGT-Netz-Interface* anzusprechen und dadurch, die Ressource einzubinden. Der Webserver dient dazu, Client-Anwendungen, die einen Zugriff auf das EFGT-Netz-Interface benötigen, im Internet bereitzustellen. Ein Beispiel hiervon ist die Webanwendung, die das Navigieren in der Struktur sowie das Editieren eines Netzes ermöglicht und Teil der Infrastruktur des Ontologieentwicklers ist (s. Abschnitt 2.2.2, S. 45 ff).

Die im Rahmen dieser Dissertation entwickelten Programme bauen auf dieser Architektur auf, sodass im Laufe der Arbeit Bezug auf obige Komponenten genommen wird und ggf. weitere Details besprochen werden.

2.1.4 Das CoGE-Netz

Das *CoGE-Netz* ist ein Beispiel einer nach dem EFGT-Formalismus realisierten Wissensressource und wird von Brunner (Brunner, 2008, Kap. 8) ausführlich beschrieben. Das Akronym steht in diesem Fall für **Co***mmon* **G***eneral* **E***ducation* und bezieht sich auf den Wissensbereich, der von diesem speziellen EFGT-Netz abgedeckt werden soll: Das breite Spektrum der Allgemeinbildung oder anders ausgedrückt, allgemeines, enzyklopädisches Wissen. Als Referenz dienen aktuelle Ausgaben deutscher Printmedien aus dem Internet.

Das CoGE-Netz hatte zu Beginn der vorliegenden Arbeit rund 60.000 Konzepte. Die unmittelbar unter dem Topnode hängenden Konzepte, die sogenannten Topkategorien, können nach deren verschiedenen Typen betrachtet werden. Die meisten der Topkategorien stellen thematische Gebiete dar und haben den Typ `F`. Diese sind in Tabelle 2.1 aufgeführt.

1. Politik
2. Wirtschaft
3. Finanzen
4. Recht und Justiz
5. Wissenschaft und Technik
6. Kunst, Kultur, Musik
7. Medien, Information und Kommunikation
8. Bildung, Erziehung und Ausbildung
9. Sport
10. Religion und Weltanschauung
11. Freizeit, Abenteuer, Lifestyle, Unterhaltung, Hobby
12. Gesundheit
13. Zivilisation, Mensch, Natur und Umwelt

Tabelle 2.1: Thematische Topkategorien des CoGE-Netzes

Den Einstieg in die zeitliche Achse bildet die Topkategorie des Typs `T` *zeitliche Gliederung/Geschichte*. Die geographische Teilhierarchie wird über die einzige Topkategorie des Typs `G`, *Welt und geographische Gliederung* erreicht. Drei Topkategorien stellen Klassen von Entitäten dar (Typ `E`): *Personen*, *Ereignisse* und *Organisationen, Institutionen und Einrichtungen*.

Von der linguistischen Perspekive her ist das CoGE-Netz multilingual angelegt, wobei die Hauptsprache Deutsch ist. Die linguistische Repräsentation der Konzepte ist ebenfalls in dieser Sprache am reichsten. 80% der Konzepte haben außerdem eine linguistische Repräsentation für das Englische, sowie jeweils 30% der Konzepte haben linguistische Angaben auf Französisch und Bulgarisch.

Die Erfahrungen, die beim Aufbau des CoGE-Netzes gesammelt wurden, bilden den Ausgangspunkt bei der Definition von Zielen für diese Dissertation. Auf diese Erfahrungen wird im zweiten Teil des Kapitels eingegangen.

2.1.5 Anwendungen

Die Hauptmotivation für die Entwicklung des EFGT-Formalismus' und den Aufbau eines EFGT-Netzes ist, eine grundlegende Wissensressource wie das CoGE-Netz für vielfältige Anwendungen der computerlinguistischen Textanalyse und Verarbeitung anzulegen. Schulz und Weigel (vgl. Schulz and Weigel, 2003) und Brunner (vgl. Brunner, 2008) nennen eine Reihe denkbarer Anwendungen. Exemplarisch seien hier folgende erwähnt:

Semantische Suche auf Dokumentenarchiven: Bei der Implementierung einer Suchanwendung auf Dokumentensammlungen kann das EFGT-Netz dafür eingesetzt werden, bei der Vorbearbeitung der Texte wichtige semantische Zusammenhänge zu erkennen. Werden Anfragen auf der Grundlage von Konzepten des Netzes formuliert, so lassen sich bei der Bestimmung der zu einer spezifischen Anfrage relevanten Dokumentenmenge die vorab ermittelten semantischen Zusammenhänge heranziehen. Dies stellt eine Alternative zur klassischen, stichwortbasierten Volltextsuche dar.

Informationsextraktion: Bei der Extraktion und Formalisierung von Information aus natursprachlichen Daten kann ein EFGT-Netz sehr nützlich sein, etwa bei der Erkennung von relevanten Entitäten sowie bei der Bereitstellung von temporalen und geographischen Zusammenhängen zur Disambiguierung von gefundenen Daten, usw.

Semantisches Tagging von Dokumenten: Unter semantischem Tagging wird die Annotation von Textdokumenten mit semantischen Informationen verstanden, z.B. mit einer vordefinierten Kategorie, mit im Text enthaltenen, besonders wichtigen Begriffen oder im Allgemeinen mit Informationen, die vom Inhalt des Dokuments abhängen. Mit Hilfe eines EFGT-Netzes können im Dokument linguistisch repräsentierte Konzepte erkannt und als Grundlage zur Ermittlung semantischer *Tags*, wie eines Themas, für das Dokument verwendet werden.

2.2 *Ontology Engineering* für EFGT-Netze

Nachdem die Grundzüge des EFGT-Formalismus' und die Motivationen für die Entwicklung von EFGT-Formalismen vorgestellt wurden, beschäftigt sich dieser Abschnitt mit dem Aufbau von EFGT-Netzen. Vor dem Hintergrund der konkreten Vorgehensweise beim Aufbau des CoGE-Netzes und der dabei gesammelten Erfahrungen werden dann im nächsten Schritt Ziele für die vorliegende Arbeit definiert.

Zunächst wird das CoGE-Netz als Ontologie-Entwicklungsprojekt charakterisiert, um anschließend die Vorgehensweise und die Infrastruktur zu beschreiben, mit denen dieses Projekt umgesetzt wird.

2.2.1 Das CoGE-Netz als Ontologieprojekt

Eine Charakterisierung des CoGE-Netzes als Ontologieprojekt lässt sich anhand der sechs Parameter von Hepp et al. (s. S. 9) bewerkstelligen.

Das CoGE-Netz ist an eine große Nutzergemeinschaft adressiert, da ein breites Spektrum der Allgemeinbildung abgedeckt werden soll und dadurch alle Nutzer angesprochen werden, die Anwendungen in diesem "Wissensbereich" benutzen.

Bezüglich seiner Ausdrucksstärke ist der EFGT-Formalismus in den ersten Stufen der Semantischen Treppe (s. S. 5) einzuordnen. Im Gegensatz zu den meisten leichtgewichtigen Ontologien, die oft mit Formalismen und Sprachen implementiert werden, die ursprünglich für andere Zwecke als die Kodierung von Ontologien entworfen wurden, beruhen EFGT-Netze auf der ID-String-Systematik. Diese kodiert Beziehungen und Eigenschaften von Konzepten explizit und kann daher als "echte" Ontologiesprache betrachtet werden. Der Formalismus ermöglicht zum einen eine explizite Typisierung der Konzepte nach dem EFGT-Prinzip, zum anderen die Definition zweier Relationen, nämlich die Nachfahre- und die Kind- oder direkte Nachfahre-Relation, die mit Hilfe des `&`-Operators und der lokale Einführung kodiert werden. Während die Nachfahrerelation im Allgemeinen sich zufriedenstellend als thematische Assoziation zweier Konzepte interpretieren lässt, wirkt die direkte Elternteil-Kind-Relation semantisch betrachtet überladen. Dies wird bei der Betrachtung verschiedener thematischer Bereiche und Typkombinationen der in Beziehung stehenden Konzepte deutlich: Mal ist das Kind als Element der durch den unmittelbaren Vorfahren ausgedrückten Menge zu lesen, mal lässt sich das Elternteilkonzept als bestimmtes Attribut des Kindkonzeptes auffassen – etwa der Beruf einer bestimmten Person –, usw.. Der

Formalismus bietet jedoch andere Modellierungsmittel, die nicht in allen leichtgewichtigen Formalismen eine Selbstverständlichkeit sind. So können Konzepte mehrere Eltern haben, d.h. die durch den Formalismus kodierte Struktur hat einen beliebigen Verzweigungsgrad. Außerdem stellt diese Struktur einen wohlfundierten Navigationsraum beliebiger Tiefe dar. Wie bereits erwähnt, ist die Ausdrucksstärke des EFGT-Formalismus' eine bewusste Wahl, mit der sich selbst in Ermangelung eines präzisen semantischen Modells thematische Assoziationsketten in vielen Bereichen modellieren lassen. Ein Vergleich mit Beschreibungslogiken und anderen Ontologieformalismen ist bei Brunner et al. (vgl. Brunner et al., 2006) zu finden.

Auf der konzeptuellen Ebene ist beim CoGE-Netz eine hohe Dynamik zu erwarten. Da das Ziel ist, ein möglichst breites Spektrum an Allgemeinwissen zu erfassen, müssen fortlaufend technische Neuerungen, sich ergebende Ereignisse, neue Personen des öffentlichen Interesses, aktuelle Themen, Änderungen, usw. berücksichtigt werden. Das zwingt dem Netz eine *open world assumption* auf: Es ist nicht davon auszugehen, dass alle existierenden Konzepte und Entitäten im Netz erfasst sind. Eine hohe Dynamik schränkt auch den Grad der Detailliertheit ein, mit der die Kodierung der einzelnen Elemente stattfinden kann. In einem feingranularen semantischen Modell können Veränderungen sehr leicht Konflikte hervorrufen und es ist unwahrscheinlich, dass die Ontolgoie monoton wachsen kann. Die erwartete Dynamik lässt selbst bei einer geringen Detailliertheit einen hohen Wartungsaufwand voraussagen, um die Ontologie aktuell zu halten.

Wegen des breit angelegten Spektrums und des holistischen Ansatzes des CoGE-Netzes ist auf der konzeptuellen Ebene ebenfalls von einer sehr großen Anzahl von Elementen auszugehen. In dieser Hinsicht ist die Skalierbarkeit der dem EFGT-Formalismus zugrunde liegenden Deduktionsmechanismen ein deutlicher Vorteil. Die große Anzahl der Elemente kann trotz des gerichteten Navigationsraumes zu Orientierungsproblemen bei den Benutzern und Problemen bei der Visualisierung des Netzes führen. Ein anderer Aspekt ist die Akzeptanz der Ontologie: Auch wenn in der Regel kleine Ontologien schneller von Nutzerkreisen angenommen werden, kann die Größe eventuell dadurch kompensiert werden, dass die einzelnen Einträge jeder für sich genommen gerade wegen seiner semantisch wenig differenzierenden Modellierung akzeptabel wirken.

In dieser Hinsicht begünstigt ebenfalls die begrenzte Spezifikationsgröße der konzeptuellen Elemente, die im Durchschnitt zwischen zwei und drei Komponenten beträgt, die Bildung von Konsens und es kann verworfen werden, dass sie zum maßgeblichen Faktor für den Kodierungsaufwand wird.

Der Grad der Subjektivität der Konzeptualisierung ist gerade bei allgemeinen Themen wie etwa Politik oder Wirtschaft als hoch einzustufen, auch wenn sicherlich Bereiche vom CoGE-Netz abgedeckt werden, bei denen sich eine allgemein geteilte Konzeptualisierung leichter finden lässt, wie etwa im geographischen oder temporalen Bereich. In den Bereichen, in denen die Konzeptualisierung subjektiv ist, ist davon auszugehen, dass eine leichtgewichtige semantische Modellierung leichter angenommen wird.

2.2.2 Technische Infrastruktur für den Ontologie-Entwickler

Brunner ermittelt in seiner Doktorarbeit (vgl. Brunner, 2008), in deren Rahmen die grundlegende technische Infrastruktur für Aufbau, Pflege und Anwendung von EFGT-Netzen implementiert wurde, die von der Rolle des Netzentwicklers gestellten Anforderungen. Der Netzentwickler lässt sich in diesem Fall weitgehend mit dem Ontologieentwickler gleichsetzen. Diesen Anforderungen wird dort mit einer Reihe von Tools begegnet, die die Arbeitsumgebung des Ontologieentwicklers im Lebenszyklus eines EFGT-Netzes darstellen und gleichzeitig den Ausgangspunkt dieser Arbeit bilden.

Laut Brunner (vgl. ebd.) stellt die Gruppe der Netzentwickler folgende Anforderungen an die technische Infrastruktur:

1. Überblicksgewinnung über Wissensstrukturen innerhalb des EFGT-Netzes, sodass ein Eindruck von den abgedeckten Wissensgebieten sowie der Struktur des Netzes entsteht. Dabei soll sowohl die Strukturierung innerhalb eines thematischen Bereiches als auch die Vernetzung der unterschiedlichen Bereiche untereinander vermittelt werden.

2. Aufbau der Ressource durch Erzeugen und Einfügen von Einträgen;

3. Editieren bestehender Einträge;

4. Editieren der inneren Struktur der Wissensressource.

Zur Überblicksgewinnung über die Ressource stehen dem Netzentwickler zum einen eine *Weboberfläche* zur Verfügung und zum anderen das sogenannte *Atlas-Tool*. Die *Weboberfläche* ermöglicht die Navigation in der DAG-Struktur der Netzes, wobei zu einem bestimmten Eintrag jeweils die berechneten Eltern und Kinder angezeigt werden. Innerhalb der Weboberfläche ist ebenfalls möglich, nach Einträgen zu suchen, die einen bestimmten Teilstring enthalten, wodurch der Einstieg in die Navigation erleichtert wird und ein Überblick über – rudimentär – linguistisch verwandte Einträge gewonnen werden kann.

Das *Atlas-Tool* dient der Verdeutlichung der Struktur eines gewählten Teils des Netzes. Dafür wird eine Linearisierung der Netzstruktur unterhalb eines bestimmten Eintrags erzeugt, ohne jedoch Blätter-Einträge mit einzuschließen. Diese Ansicht ist zum Ausdrucken gedacht, wobei die Einträge eine spezielle Nummerierung erhalten, die das Hin- und Herblättern erleichtern soll. Taxonomische Beziehungen werden durch Einrückung dargestellt.

Das Anlegen eines neuen Eintrags erfolgt mittels eines *Eintragsformulars*, das in der Weboberfläche integriert ist. Im Formular wird zunächst ein neuer ID-String angegeben, wodurch die Position des neuen Eintrags in der vorhandenen Netzstruktur ermittelt und für die Navigation in der Weboberfläche erreichbar gemacht wird. Anschließend können im Formular der Namen des Eintrags sowie zusätzliche Daten abgegeben werden.

Das Editieren bestehender Einträge erfolgt ebenfalls mittels des Formulars. Da das direkte händische Editieren eines ID-Strings je nach Länge und Verschachtelung wegen der schlechten Lesbarkeit sehr fehleranfällig ist, wurde ein spezieller Editor hierfür implementiert. Dieser sogenannte *Baumeditor* parst den ID-String und stellt ihn als Baum von

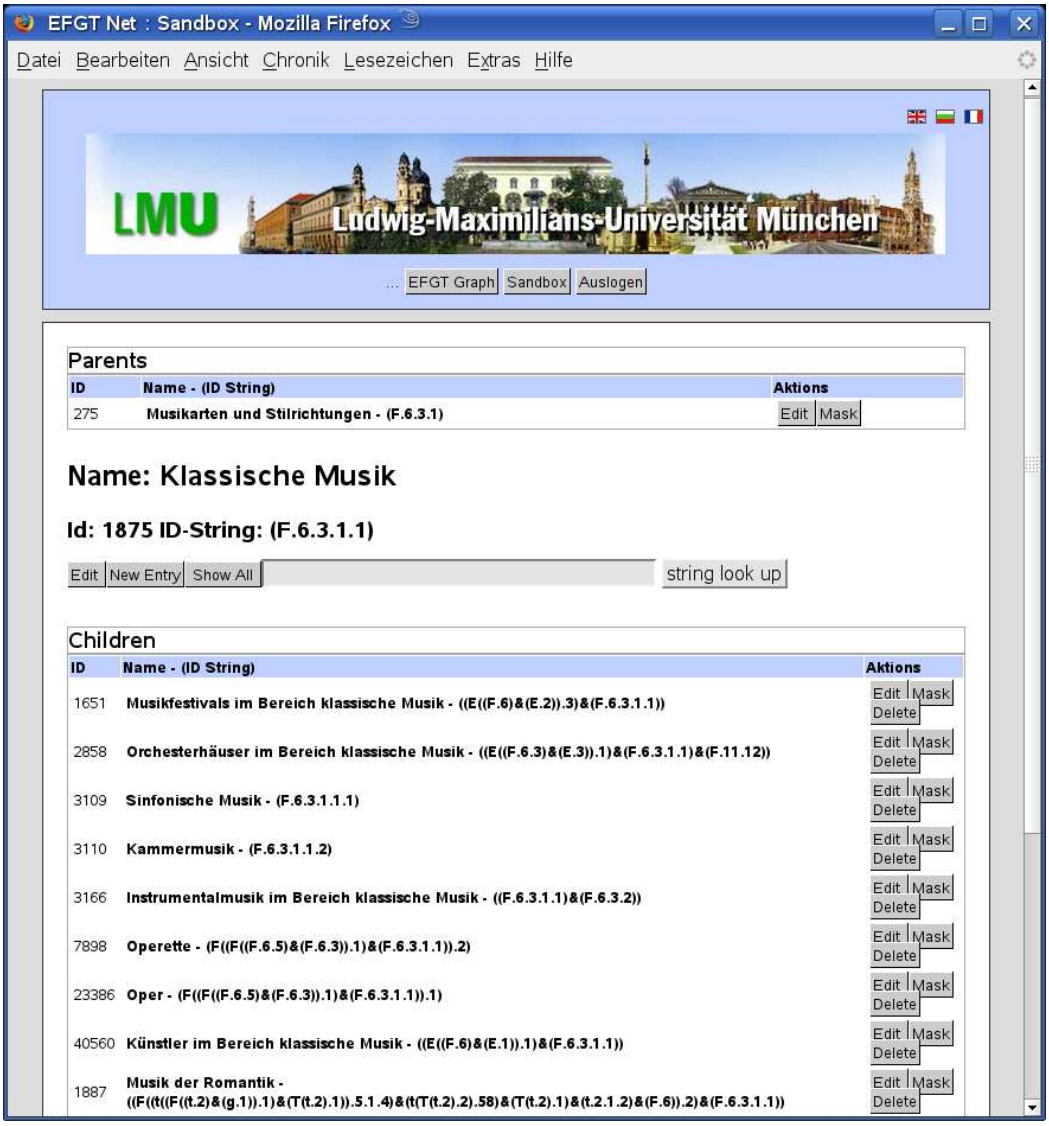


Abbildung 2.2: Weboberfläche zur Graphdarstellung und Navigation in einem EFGT-Netz

Teilkomponenten dar. Dadurch wird der Aufbau des Strings visualisiert und der semantische Gehalt des Eintrags aufgeschlüsselt, sodass die Manipulation des ID-Strings wesentlich erleichtert wird.

Das Editieren der inneren Struktur der Ressource erfolgt auf der Ebene einzelner Einträge durch das Editieren des entsprechenden ID-Strings im Formular, wobei hierfür sich der Baumeditor zur Hilfe heranziehen lässt. Das Löschen einzelner Einträge erfolgt in der Navigationsansicht der Weboberfläche über entsprechende *Delete*-Knöpfe (s. Abb. 2.2). Oft will man aber einen Eintrag durch Editieren des ID-Strings umhängen und den Effekt an alle seine Nachfolger weitergeben, die den modifizierten ID-String als Komponente enthalten. Ein solches Umhängen eines ganzen Astes des Netzes kann mit Hilfe des *Rename-*

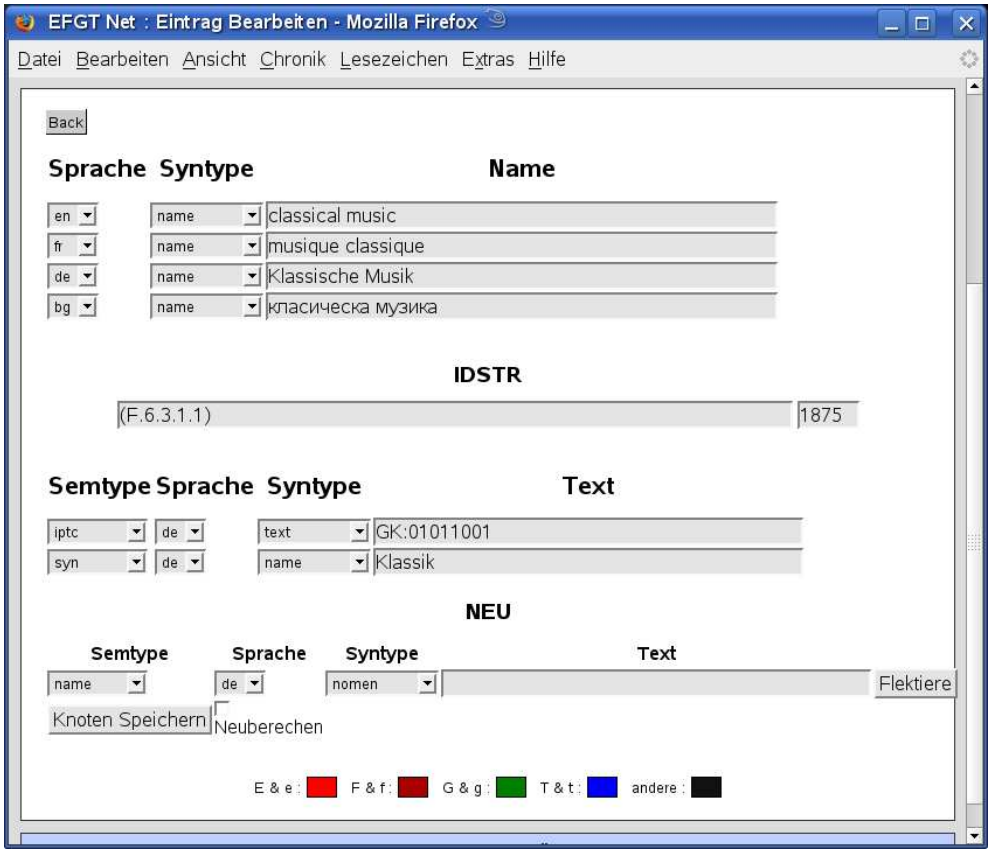


Abbildung 2.3: Beispiel eines EFGT-Netz-Eintrags im Eintragsformular

Tools durchgeführt werden, das dem Netz-Entwickler das händische Editieren aller von der Änderung betroffenen ID-Strings erspart. Zunächst wird eine Vorschau aller Änderungen erzeugt, die sich durch Ersetzen des modifizierten ID-Strings in allen ihn als Teilkomponente enthaltenden ID-Strings ergeben. Dem Benutzer wird hiermit die Möglichkeit gegeben, automatisch nicht auflösbare Konflikte händisch zu bearbeiten. Anschließend wird ein Skript erzeugt, mit dem die Änderungen in der Datenbank tatsächlich durchgeführt werden können.

Weitere Aspekte der Software-Infrastruktur, die die Anforderungen der technischen Entwickler und Anwender betreffen, wie etwa unterschiedliche Schnittstellen und Anbindungsmöglichkeiten, werden im Zusammenhang mit den im Laufe dieser Arbeit entwickelten Programme vorgestellt.

2.2.3 Der Lebenszyklus eines EFGT-Netzes am Beispiel des CoGE-Netzes

Abbildung 2.5 zeigt eine schematische Darstellung der Vorgehensweise beim Aufbau des CoGE-Netzes. In folgendem Abschnitt werden die verschiedenen Phasen in diesem Le-

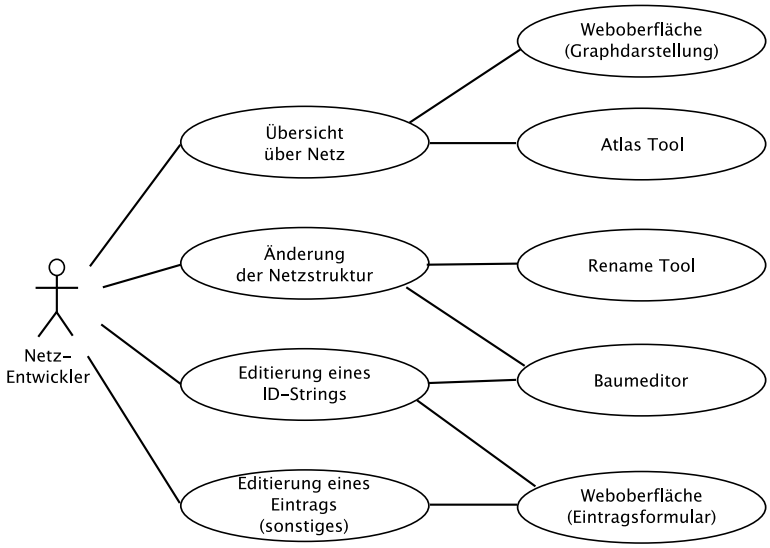


Abbildung 2.4: Generische und praktische Anwendungsfälle für den Netzentwickler. Praktische Anwendungsfälle (links) korrespondieren mit bereitgestellten Anwendungen (Brunner, 2008, S. 46)

benszyklus beschrieben. Hierbei wird oft Bezug auf *Ontology Engineering Aktivitäten* aus Kapitel 1, S. 13 ff. genommen.

A. Wissensakquisition

Die Wissensakquisition stellt für ein breit angelegtes Ontologieprojekt wie das CoGE-Netz eine Herausforderung dar. Das liegt vorrangig an der sehr breit angelegten anvisierten Wissensdomäne und an der Tatsache, dass in dieser Phase zwei komplexe Aktivitäten ineinander übergehen, wie sich bei genauer Betrachtung herausstellt. Zum einen wird in dieser Phase implizit die Konzeptualisierung der Wissensdomäne entwickelt, die der Ontologieentwickler einzuarbeiten hat. Bei einer Ressource wie dem CoGE-Netz, das thematische Zusammenhänge zu erfassen versucht, die das Allgemeinwissen widerspiegeln, kann der Ontologieentwickler, der in diesem Fall kein Experte in einem bestimmten Wissensbereich zu sein braucht, zunächst mal vom eigenen Kenntnisstand ausgehen. Als Vorbereitung für die Kodierung vergleicht und ergänzt der Entwickler ggf. sein Wissen durch Recherche im Internet, Nachschlagen in Enzyklopädien, Referenzquellen, usw. Dadurch wird die eigene Konzeptualisierung überprüft und erweitert bzw. näher an eine allgemein geteilte Konzeptualisierung gerückt. In den meisten Fällen ist es so, dass der Ontologieentwickler diese Einsichten dann direkt in die Ressource einarbeitet, d.h. das abstrakte Modell der Konzeptualisierung wird ausschließlich durch die Kodierung dokumentiert. Die Konzeptualisierung wird in der späteren Kodierungsphase (s.u.) weiter präzisiert, sodass die Aufstellung eines abstrakten Modells sich unscharf über beide Phasen erstreckt.

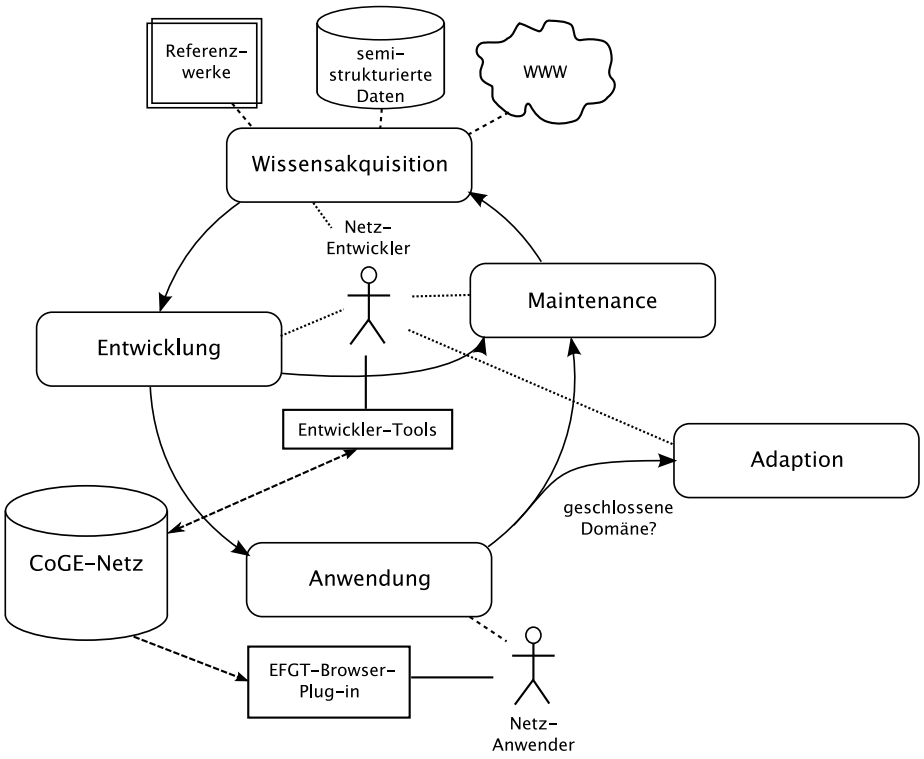


Abbildung 2.5: Phasen im Lebenszyklus eines EFGT-Netzes am Beispiel des CoGE-Netzes

Zum anderen dient die Wissensakquisition dazu, das Datenmaterial zur Verfügung zu stellen, das in der Kodierungsphase in die Ontologie eingearbeitet wird. Somit stellt diese Phase mit der Beschreibung der verschiedenen Aktivitäten auf S. 13 ff übereinstimmend eine *support activity* für die eigentliche Implementierung der Ontologie dar. Diese Daten sind unterschiedlicher Natur, je nachdem, welche Teile des Netzes es zu kodieren gilt.

Zum Aufbau eines thematischen Bereiches sind bereits existierende Taxonomien und Ontologien über den Bereich besonders nützlich. Je nach Bereich lässt sich hier auf bekannte, existierende Quellen wie etwa andere Ontologien zurückgreifen. Ansonsten kann die Wissensakquisition durch Recherchen erfolgen, die, wie oben dargestellt, den zusätzlichen Effekt haben, die Konzeptualisierungsaktivität zu unterstützen sowie themenspezifische Begriffe aus Textsammlungen zu ermitteln, die den thematischen Bereich abdecken.

Für die Population der Ontologie mit Entitäten wie Personennamen, Firmennamen, usw. ist insbesondere die Akquisition von listenartigen oder tabellarischen Daten wichtig, die Angaben enthalten, die bei der Kodierung die Rolle von Facetten übernehmen können, z.B. bei Personennamen, Beruf, Nationalität, Geburtsjahr, usw. (vgl. Kap 3, Abschnitt 3.1).

Als Grundlage für die Kodierung der geographischen Achse des Netzes lässt sich in vielen Fällen auf öffentlich verfügbare Datenbanken zurückgreifen. Die Hauptaufgabe hier besteht darin, die Daten aus diesen Datenbanken entsprechend der maßgeblichen Konzeptualisierung strukturiert abzufragen und mit Ergebnissen aus anderen Datenbanken

zusammenzuführen.

Bei dem Aufbau der zeitlichen Achse des Netzes sind vor allem Daten über unterschiedliche Epochen und Zeitintervalle relevant, die in den verschiedenen Disziplinen konzeptuell existieren, wie etwa Kunst- oder geologische Epochen. Die Akquisition findet hier meistens händisch unter Verwendung von Nachschlagewerken statt.

Unabhängig von dem anvisierten Teil des Netzes fällt ein Teil des Aufwands bei der Wissensakquisition an die Erfassung und Zusammenführung von linguistischen Varianten, mit denen auf eine bestimmte Entität oder Begrifflichkeit textuell Bezug genommen wird.

Die in dieser Phase eingesetzten technischen Mittel sind vielfältig und gehen weit über die oben beschriebene Infrastruktur für den Netzentwickler hinaus. Je nach Datenquelle werden verschiedene Techniken eingesetzt, wie Wrappertools und Informationsextraktionstechniken, Crawler zum Zusammentragen von Webdaten und Texten, Tools zur Umformatierung oder auch Methoden der Corpus- und linguistischen Analyse, hier insbesondere der Kompositaanalyse.

B. Entwicklungsphase: Kodierung des CoGE-Netzes

Mit Kodierung ist das Anlegen neuer Einträge ins EFGT-Netz gemeint, insbesondere die Ausarbeitung des zugehörigen ID-Strings zu jedem Eintrag. Die Kodierungsphase korrespondiert mit den *development activities* auf S. 13.

Die Analyse der Domäne hat durch die explizite Zielsetzung des Projektes und die Designkriterien des EFGT-Formalismus' bereits im Vorfeld stattgefunden. Zwischen den Aktivitäten der Konzeptualisierung, der Modellierung sowie der Implementierung der Ontologie mittels des EFGT-Formalismus' kann, wie oben angesprochen, keine scharfe Grenze gezogen werden. Die Konzeptualisierung wird ohnehin a priori stark beeinflusst durch die EFGT-Betrachtungsweise, die Entitäten in den durch thematische, geografische und temporale Achsen aufgespannten Raum einordnet. Ein Großteil der Kodierung findet in einer *middle-out*-Manier statt. Für eine thematisch sehr breit gestreute Ontologie wie das CoGE-Netz bedeutet dies, dass die Konzeptualisierung stark vom Zustand des Netzes zum Zeitpunkt der Konzeptualisierung beeinflusst wird, da es darum geht, neue Konzepte und Bereiche in das vorliegende Netz einzuarbeiten. Das ist so, weil der Grad der Subjektivität je nach Bereich stark variiert.

Die Kodierung eines Konzeptes folgt trotz der hohen Subjektivität der Konzeptualsierung in einem bestimmten Sinn einem analytischen Kriterium: Als Grundlage für die Definition eines ID-Strings dient der semantische Gehalt des Konzeptes, d.h. dessen Beziehungen zu anderen Konzepten in einem allgemeinen Kontext. Im Gegensatz dazu würde ein pragmatisches oder anwendungsorientiertes Kodierungskriterium die Anforderungen einer bestimmten Anwendung berücksichtigen. Bei einer Suchanwendung beispielsweise wäre dies, dass die Verlinkung des Netzes so angelegt ist, dass durch Navigation möglichst effizient zu den relevanten Dokumenten der indexierten Dokumentensammlung gelangt. Ein solches pragmatisches Kriterium bei der Implementierung einer grundlegenden, holistischen Ontologie zu verfolgen würde jedoch stark das Spektrum der Anwendungen einschränken, in die sie sich als Ressource einsetzen ließe.

Phasen der Kodierung Bei der Kodierung eines EFGT-Neztes wie dem CoGE-Netz werden verschiedene Phasen unterschieden. Am Anfang wird der Kern der Ontologie entwickelt, der das Rückgrat für die weitere Entwicklung bildet. Dafür werden die Kategorien auf der höchsten Ebene der Ontologie definiert, insbesondere die vom Typ `F`, die den thematischen Umfang der Ontologie bestimmen sowie die Grundmengen für die unterschiedlichen Entitäten, die in die Ontologie einzuordnen sind. Zum Kern zählen auch die temporale und die geographische Achse, die orthogonale Dimensionen bilden und sich unabhängig vom Rest der Ontologie gut in einer *top-down* Vorgehensweise entwickeln lassen. Der so gebildete Kern sollte möglichst stabil bleiben, weil spätere Änderungen an den oberen Ebenen der Hierarchie zu umfangreichen Änderungen an der Struktur des Netzes führen.

Von der Kodierung des Kerns wird zum Ausbau der Ressource übergegangen, indem die thematische Achse des Netzes entfaltet wird. Dies erfolgt zum einen durch Verfeinerung der vorhandenen thematischen Kategorien, etwa durch Einarbeiten einer Taxonomie. Zum anderen wird die thematische Dimension durch Facettierung ausgebaut, d.h. durch die Kombination bestehender Themen mit einer oder mehreren geographischen bzw. temporalen Komponenten zu einem neuen Thema. Dadurch entstehen Querverbindungen zwischen den verschiedenen Dimensionen und das Netz verdichtet sich. Entitäten und Entitätenmengen werden typischerweise ebenfalls durch Facettierung in das Netz eingeführt, und bauen somit auf bereits existierenden Konzepten auf. Mit dem Einsatz der Facettierung drückt sich die *middle-out*-Strategie aus, die beim Ausbau der thematischen Achse und der Kodierung von Entitäten verfolgt wird.

Als zusätzliche Phase kann man das Befüllen oder *Population* der Ontologie mit einzelnen Entitäten bezeichnen, die als Instanzen oder Blätter an einzelne Container-Konzepte gehängt werden. Bei der Kodierung dieser Instanz-Entitäten werden bestehende Konzepte als Facetten für das neue Konzept ausgewählt, das typischerweise als neuer Eintrag des Typs `e` oder `g` lokal eingeführt wird. Als Population könnte man auch den Ausbau eines sehr feingliedrigen thematischen Bereichs auffassen. Gerade in dieser Phase ist der Ontologieentwickler auf Daten aus dem Wissenakquisitionsschritt angewiesen, insbesondere auf vorab schon (semi-)strukturierte Datenquellen, die die für die Kodierung notwendige Information über mögliche Facetten enthalten.

Die Phasen der Population und des thematischen Ausbaus wechseln sich oft in einem iterativen Prozess ab.

Probleme der Kodierung Brunner (vgl. Brunner, 2008) beschreibt verschiedene Probleme, die sich bei der Kodierung ergeben können, wobei manche dieser Probleme vom EFGT-Formalismus begünstigt werden. Hierzu zählt das sogenannte Problem des *schleichenden Kontextes*, das entsteht, wenn ein Konzept unter dem Blickwinkel einer bestimmten Disziplin oder Wissensdomäne im EFGT-Netz modelliert wurde, dabei jedoch nicht beachtet wurde, dass dasselbe Konzept aus einer allgemeineren Perspektive heraus nur eine unwesentliche Zugehörigkeit zu dem Thema hat. Ein Beispiel ist, wenn *Gold* als Konzept des Themenbereichs *Finanzen* etwa als Anlagegegenstand im Rohstoffmarkt modelliert worden wäre. Gold existiert jedoch als Konzept unabhängig von der Finanzwelt und

taucht deswegen auch in anderen Kontexten auf, etwa als chemisches Element, Schmuckgegenstand, als Material in der Odontologie usw.. Sollte Gold ausschließlich als Konzept der Finanzwelt modelliert worden sein, wären wegen der Transitivität der thematischen Relation die anderen Konzepte, z.B. *Goldzahn* als auf Gold als Komponente aufbauend, ebenfalls der Finanzwelt untergeordnet – was mehr als fragwürdig wäre. Ein Ausweg aus diesem Problem besteht darin, den spezifischen Aspekt des Konzeptes, der die Modellierung in dem Themenbereich motiviert, in die Konzeptbezeichnung zu integrieren, etwa *Gold als Rohstoff*. Damit lässt sich vermeiden, dass jedes textuelle Vorkommen des Konzepts (Gold) als ein Hinweis auf den spezifischen Themenbereich (Finanzwelt) verstanden wird.

Bei langen Pfaden im Netz können sich außerdem noch *thematische Verschiebungen* ergeben, ein weiteres Problem, das eng mit dem EFGT-Formalismus verbunden ist. Dabei wird ein untergeordnetes Konzept als nicht mehr zugehörig zu einem weit entfernten Überbegriff empfunden. Zugleich kann jedoch jeder einzelne Schritt in der Kette sinnvoll und nachvollziehbar erscheinen. *München* würde als Unterbegriff von Sport als unzulässig wahrgenommen, obwohl es über dem Konzept *Austragungsstätten von Olympischen Spielen* durchaus eine sinnvolle Verbindung mit dem Thema geben kann. Eine Lösung zur Vermeidung solcher thematischen Verschiebungen ist das Einführen eines Konzeptes, das dem im Konflikt stehenden, untergeordneten Konzept als Rolle bezüglich des übergeordneten Themas dient, etwa *München als Olympiaaustragungstätte*. Durch die neue Verbindung des sonst "verschobenen" Themas mit dem Rollenkonzept statt mit dem ursprünglich Untergeordneten wirkt die thematische Einordnung korrekt. Diese Probleme fallen oft erst bei der Verwendung des Netzes auf und lassen sich nur durch eine Umstrukturierung – womöglich mit Hilfe des Rename Tools – lösen, was mit einem erheblichen Aufwand verbunden ist. Brunner (Brunner, 2008, S. 129 ff) nennt Kodierungskriterien, die die Gefahr des Auftretens dieser Probleme minimieren sollen.

Andere Kodierungsprobleme entstehen speziell bei der Kodierung einer bestimmten Achse. Beispielsweise bestehen auf der temporalen Hierarchie unter anderem Schwierigkeiten mit Zeiträumen wie bestimmten Epochen, die unscharf sind, oder die sich in unterschiedlichen Ländern zu unterschiedlichen Zeiten zugetragen haben. Ähnlich sind viele geographische Begriffe in der natürlichen Sprache vage und bezeichnen keine klar definierten Bereiche. Wegen des starken Bezugs mancher geographischen Gebiete zu einem Thema oder einer Zeitperiode, wie etwa *Reich der Pharaonen*, ist oftmals die ideale Typenzuordnung unklar.

Kodierung eines EFGT-Netzes mit Hilfe der Weboberfläche Unabhängig vom Ontologiebereich erfolgt die Kodierung händisch mit Hilfe der oben beschriebenen Weboberfläche und dem Webformular. Der Aufwand für einen einzelnen Eintrag entsteht hierbei hauptsächlich durch die Vorbereitungen, die für die Realisierung des Eintrags vorgenommen werden. Neben dem Aufwand der Wissensakquisition muss der Ontologieentwickler entweder durch Suche oder durch Navigation passende semantische Komponenten, die der erarbeiteten Konzeptualisierung entsprechen, im Netz identifizieren und sammeln. Das ist oft ein aufwändiger Prozess, da wegen der Breite des Netzes der Entwickler nicht immer

mit dem Thema vertraut ist und sich verschiedene Möglichkeiten anbieten können. Es kann auch leicht passieren, dass der neue Eintrag semantisch nicht gut vorbereitet ist und zusätzliche Konzepte eingeführt werden müssen, um eine zufriedenstellende Modellierung des neuen Eintrags erreichen zu können. In vielen Fällen bedeutet ebenfalls die Angabe der einzelnen Flektionsformen und Varianten einen händischen Aufwand.

Ein Problem ist auch die große Anzahl der Einträge, die gerade in der Phase der Population vorzunehmen sind und einzeln im Webformular bearbeitet werden müssen. Oft hängen die neuen Einträge zusammen und müssen in einer bestimmten Reihenfolge bearbeitet werden, beispielsweise dann, wenn es darum geht, eine Taxonomie in das Netz zu integrieren. Die Planung dieser Einträge muss außerhalb der Weboberfläche stattfinden, da sie keine Unterstützung für das Einhalten einer bestimmten Eintragsreihenfolge bietet.

C. Anwendung

Wie in Abschnitt 2.1.5 beschrieben, stellen die semantische Annotation (Indexierung) von Webdokumenten und Dokumentenarchiven auf der Basis eines EFGT-Netzes das grundlegende Verfahren dar, dass Anwendungen wie semantisches Retrieval, *typed hyperlinking*, semantisches Browsen, usw. ermöglicht. Die Erfahrungen im Anwendungsbereich von EFGT-Netzen beruhen bisher auf zwei Systemen, nämlich dem EFGT-Netz-Browser-Plug-In (vgl. Weigel et al., 2006) und einer semantischen Browsinganwendung für Dokumentenarchive. Letztere ist allerdings erst im Rahmen dieser Arbeit entstanden (vgl. Kap. 4) und stand bei der Entwicklung eines wesentlichen Teils des CoGE-Netzes noch nicht zur Verfügung.

Mit dem EFGT-Netz-Browser-Plug-In kann in einem Webbrowser eine Ansicht des EFGT-Netzes integriert werden, sodass neben dem normalen Anzeigefenster für das Webdokument die Navigation im EFGT-Netz möglich wird. Lädt man dann wie üblich ein Webdokument in den Browser, so werden im Dokument die Konzepte markiert, die aus dem EFGT-Netz bekannt sind. Dieser Schritt der semantischen Annotation bildet die Grundlage für zusätzliche Funktionen im Browser, die eine Einschätzung des semantischen Gehalts des aktuell angezeigten Dokuments ermöglichen: Es können dann etwa die Treffer eines bestimmten Typs im Dokument hervorgehoben werden; durch Klicken auf einen Treffer kann auf die entsprechende Stelle des EFGT-Netzes zurückgesprungen werden; ebenfalls können im Dokument die Treffer hervorgehoben werden, die im EFGT-Netz in Verbindung mit dem in der Navigationsleiste aktuell ausgewählten Konzept stehen; die Distribution der Dokumententreffer im Netz wird visualisiert, usw.

Über einen längeren Abschnitt der Entwicklung des CoGE-Netzes wurde das EFGT-Netz-Browser-Plug-In mit einer Kontrollfunktion eingesetzt. Durch die bereitgestellten Funktionen entstand ein Kontext, in dem sich in Bezug auf einzelne Dokumente für Teile des Netzes die letztendliche Qualität der entwickelten Konzeptualisierung und der analytischen Vorgehensweise bei der Kodierung ansatzweise zeigen konnte. Dadurch konnten Dokumente sowohl einzelne Fehler der Kodierung als auch inhärente Probleme der Modellierung mit dem EFGT-Formalismus sowie Ambiguitäten entdeckt werden. Als weitere Qualitätssicherungsmaßnahme wurde regelmäßig mit Hilfe der Weboberfläche im CoGE-Netz themenbezogen navigiert und dabei das Netz von den Kodierern händisch überprüft.

Eine Methode zur automatisierten Evaluation des Netzes (vgl. Kap. 1, Abschnitt 1.3.4), d.h. von einer pragmatischen Perspektive aus, ist weder definiert noch regelmäßig angewendet worden.

D. Pflege und Maintenance bzw. Adaption

Das Beheben von Problemen auf der Modellierungsebene, die etwa im Rahmen einer Anwendung des EFGT-Netzes entdeckt wurden, ist eine der Funktionen der regelmäßigen Pflege des Netzes (*Maintenance*). In dieser Hinsicht ist die Maintenance ein Teil des iterativen Entwicklungskreises im Lebenszyklus eines EFGT-Netzes. Beim CoGE-Netz stellt die Maintenance eine Aktivität dar, die der Kodierung sehr ähnlich ist und die zu Beginn dieser Arbeit mit derselben technischen Infrastruktur erfolgt ist.

Die allgemeine Gefahr der Maintenance besteht darin, dass, motiviert durch eine bestimmte Anwendung wie das EFGT-Netz-Browser-Plug-In Änderungen an der Kodierung vorgenommen werden, die sich nachträglich als nicht intendierte Anpassungen an ebendiese Anwendung herausstellen, sodass die Ressource sich nicht mehr für den allgemeinen Einsatz in unterschiedlichen Anwendungen eignet. In diesem Fall hätte man das Prinzip des *minimal ontology commitment* verletzt, indem man sich an einem pragmatischen Kriterium orientiert.

Unabhängig von einer bestimmten Anwendung ist ein weiteres Ziel regelmäßiger Pflege des Netzes die längerfristige Qualitätssicherung der Ressource in Bezug auf die Zieldomäne. Je nach Dynamik der Zieldomäne auf der Konzeptualisierungsebene kann man grob zwei Fälle unterscheiden:

- Eine *offene Wissensdomäne*, in der immer wieder neue Konzepte entstehen. Aus der Perspektive des EFGT-Netzes korrespondiert eine offene Domäne mit einer offenen Dokumentensammlung, in der immer wieder neue Dokumente hinzukommen und die sich außerdem nicht auf ein bestimmtes Fachgebiet beschränken. Zu einer solchen thematisch offenen Dokumentensammlung kommen also immer wieder neue Themen und Entitäten hinzu, aber auch ergänzende Informationen zu bereits bestehenden, die eine Änderung oder Verfeinerung ihrer Modellierung anregen. Auf der Ebene des Allgemeinwissens ist das Ziel des CoGE-Netzes, eine solche thematisch offene Wissensdomäne zu erfassen. In einem Netz wie diesem muss notgedrungen eine *open world assumption* gelten.

- Eine *geschlossene Wissensdomäne*, in der die Liste der relevanten Konzepte endlich und fest bleibt. Eine solche Domäne wird entweder durch eine zwar variable, aber sich auf einen wenig innovativen Fachbereich beschränkende Dokumentensammlung – eher die Ausnahme – oder durch eine geschlossene Dokumentensammlung repräsentiert, auch wenn diese im Prinzip thematisch sehr vielfältig sein kann.

Die Dynamik der vom CoGE-Netz anvisierten, offenen Wissensdomäne macht eine fortlaufende Pflege des Netzes notwendig, um auf Dauer seine Qualität zu erhalten. Bei einer Methodologie, wie sie bei der Entwicklung des CoGE-Netzes verfolgt wurde und bei der der

Kreis der Ontologieentwickler getrennt vom Anwenderkreis existiert, bedeutet eine fortlaufende Maintenance auch eine fortlaufende, vorausschauende Wissensakquisition zusammen mit einer fortlaufenden (Weiter-)Entwicklung des Netzes und somit einen signifikanten Aufwand. Eine Alternative hierzu wäre, die Nutzer einer bestimmten Anwendung am Prozess der Maintenance zu beteiligen, etwa, indem ihnen die Behebung von Fehlern und die Weiterentwicklung der Ressource überlassen wird. In einem solchen Szenario ist es jedoch schwer, die allgemeine Maintenance, in der die Kodierung nach einem analytischen Kriterium stattzufinden hat, von der Anpassung an die Anwendung nach einem pragmatischen Kriterium zu trennen.

Handelt es sich bei der anvisierten um eine geschlossene Domäne im obigen Sinne, lassen sich unter dem Gesichtspunkt der Maintenance zwei Fälle unterscheiden. Im Falle eines Textkorpus', der sich sehr nach einem bestimmten Wissensbereich richtet, kommt bei der Realisierung von Anwendungen der Einsatz eines spezialisierten EFGT-Netzes eher in Frage als ein CoGE-Netz, das einzelne Bereiche weniger detailliert modelliert. Die Maintenance würde in diesem Fall, solange keine anwendungsspezifischen Änderungen übernommen werden, zur Qualität des spezialisierten Netzes beitragen. Es ist davon auszugehen, dass nach einer endlichen Anzahl von Iterationen des Lebenszyklus die Qualität der Ontologie endgültig gesichert und der Prozess der Maintenance im Wesentlichen als abgeschlossen betrachtet werden kann. Hierbei scheint es wegen des starken Bezugs auf einen definierten Wissensbereich möglich zu sein, keine anwendungsspezifischen Modifikationen einzuführen und die Wiederverwendbarkeit der Ontologie für andere Anwendungen zu wahren.

Im anderen Fall, dem einer zwar geschlossenen, jedoch thematisch breit gestreuten Dokumentensammlung, erscheint der Einsatz des CoGE-Netzes dagegen durchaus sinnvoll. Als spezialisierte Ressource für diese Sammlung lässt sich das Teilnetz auffassen, das die in den einzelnen Dokumenten behandelten Themen abdeckt. Bei einer geschlossenen Dokumentensammlung sind aus der Sicht spezifischer Anwendungen thematische Lücken jedoch potentiell problematischer als bei einer thematisch offenen. Je nach Streuung und Vielfältigkeit der darin repräsentierten Wissensbereiche kann es nötig sein, weitere in der Dokumentensammlung vorkommende Konzepte in das Netz aufzunehmen, um einen geeigneten Detaillierungsgrad der Ressource zu erreichen. An dieser Stelle ergibt sich die Frage, nach welchem Kriterium die Kodierung der in der Dokumentensammlung neu entdeckten Konzepte stattfinden soll.

Zunächst würde man die neuen Konzepte aus der anwendungsunabhängigen Perspektive des analytischen Kriteriums kodieren, sodass sie zum Ausbau der allgemeinen Ontologie, des CoGE-Netzes, beitragen und weiteren potentiellen Anwendungen ebenfalls zugute kommen. Oft stellt sich in der Praxis heraus, dass die aufzunehmenden Konzepte durch die Betrachtung spezieller Kontexte für eine bestimmte Anwendung eine wichtige Rolle spielen, auch wenn von einem allgemeinen Blickwinkel aus betrachtet dies nicht der Fall wäre (vgl. S. 51, Probleme der Kodierung; schleichender Kontext). Hier würde man das analytische Kodierungskriterium aufgeben, um ein pragmatisches, anwendungszentriertes zu verfolgen und statt der Maintenance die *Adaption* des relevanten Teils des Netzes an die spezifische Dokumentensammlung vornehmen. Die damit entstehende spezialisierte Ressource wäre dann bewusst nicht zur Wiederverwendung gedacht.

Mit der hier beschriebenen verfügbaren technischen Infrastruktur lässt sich die Adaption des Netzes durchführen, indem ein relevantes Teilnetz etwa durch die Indexierung der Dokumentensammlung ermittelt und in eine separate Datenbank ausgelagert wird. Zur tatsächlichen Adaption wird anschließend dieses Teilnetz mit Hilfe der Weboberfläche weiterentwickelt. Dadurch entsteht eine unabhängige, anwendungsspezifische Ressource mit einem eigenem Lebenszyklus.

In der Regel geht die Adaption über die Kodierung zusätzlicher Konzepte hinaus, mit denen die angemessene Abdeckung der betrachteten Dokumentensammlung sichergestellt wird: Durch die Einschränkung auf bestimmte Themen werden beispielsweise bestimmte Abstraktionsebenen der Hierarchie in der spezialisierten Ressource überflüssig, sodass es sich anbietet, sie dementsprechend anzupassen. Auch ist es meistens sinnvoll, Ambiguitäten soweit wie möglich händisch aufzulösen, um irrelevante thematische Teilbereiche auszuschließen, die in der betrachteten Domäne zu einer Fehlinterpretation der Konzepte mit ambigen Bezeichnern führen würden.

Durch die Adaption der Ressource an die Anwendung kann diese profitieren oder überhaupt erst realisierbar sein. Dass ein solcher Schritt unter Umständen nötig ist, könnte einerseits als ein Nachteil der analytischen, von der Anwendung unabhängigen Vorgehensweise bei der Kodierung der Ressource aufgefasst werden. Andererseits muss das analytische Kriterium bewahrt werden, wenn nicht ein neues Netz für jede neue Anwendung entwickelt werden soll. Stünden geeignete Methoden zur Verfügung, die eine einfache Adaption der Ressource an diverse Anwendungen ermöglichen würden, ließe sich sowohl der Aufwand dieser Phase minimieren als auch die Neuentwicklung anwendungsspezifischer Ressourcen vermeiden.

Die vorangehende Diskussion zeigt, dass eine Trennung zwischen universeller und spezifischer, anwendungsoptimierter Ontologie schwer zu wahren ist. In jedem Fall liefert der Einsatz innerhalb einer Anwendung wesentliche Informationen für die Pflege der Ontologie; bei der Behebung dieser Probleme muss daher eine bewusste Entscheidung zwischen Adaption und Maintenance getroffen werden, um die Qualität der Ressource zu sichern.

2.2.4 Schlussfolgerungen

In diesem Abschnitt wurde die Vorgehensweise vorgestellt, die bei der Entwicklung eines bestimmten EFGT-Netzes, nämlich des CoGE-Netzes, verfolgt wird. Diese Beschreibung hat weniger den Anspruch, *best practices* darzustellen als zu vermitteln, wie EFGT-Netze in der Praxis tatsächlich aufgebaut werden. Diese Vorgehensweise weist viele Gemeinsamkeiten mit anderen, im Kapitel 1 vorgestellten Methodologien auf.

Bei der Entwicklung des CoGE-Netzes wird wie üblich eine *evolving prototype* Strategie verfolgt, in der jederzeit Änderungen an der Ressource vorgenommen werden können und bei der eine klare Trennung zwischen dem Kreis der Ontologieentwickler und dem Kreis der Ontologieanwender gegeben ist. In der vorgestellten Vorgehensweise durchläuft das CoGE-Netz verschiedene Phasen, die vergleichbar mit dem Lebenszyklus der aktuelleren Methodologien aus Kapitel 1 sind. Insbesondere wird eine Endanwendung, nämlich das EFGT-Netz-Browser-Plug-In, eingebunden, mit dem zumindest ein rudimentäres Bild

über den Entwicklungszustand der Ressource gewonnen werden kann. Allerdings werden Fehler und Verbesserungsvorschläge, die sich während der Endanwendung ergeben und später in der Maintenancephase genutzt werden können, in keinem organisierten Prozess systematisch erfasst. Als ergänzende Qualitätssicherungsmaßnahme findet eine regelmäßige manuelle Überprüfung der Ontologie durch die Kodierer selbst statt. Ein Merkmal des Lebenszyklus' des CoGE-Netzes ist die fortlaufende Maintenance der Ressource, die sich kaum von einer fortlaufenden Entwicklung unterscheiden lässt. Wegen der Breite und Dynamik des anvisierten Wissensbereiches ist diese fortlaufende Entwicklung mit einem erheblichen Aufwand bei der Wissensakquisition verbunden. Ein weiteres Merkmal ist das Fehlen eines definierten Evaluationsschrittes. Da Aktivitäten in den einzelnen Schritten der Methodologie wenig operationalisiert sind, hat sie im aktuellen Stadium insgesamt einen experimentellen Charakter. Dies zusammen mit der Offenheit des Entwicklungsprozesses kann als Grund angesehen werden, dass insgesamt keine Management-Aktivitäten stattfinden.

Aus technologischer Sicht fällt bei der vorgestellten Methodologie auf, dass bestimmte Anforderungen, die sich erst während des Durchlaufens der unterschiedlichen Phasen zeigen, von der ursprünglich bereitgestellten technischen Infrastruktur (s. Abschnitt 2.2.2) nicht abgedeckt werden:

- Bei der Wissensakquisition wird auf bereits existierende Daten, Tools und Methoden zurückgegriffen. Die Integration der dadurch gewonnenen Daten wird jedoch technisch nicht unterstützt. Das trifft sowohl auf die Integration von externen Hierarchien im EFGT-Netz als auch auf die Population mit facettierten Daten zu.

- Ein kollaboratives Szenario bei der Ontologie-Entwicklung wird zwar durch die Weboberfläche unterstützt. Kommentare können den Einträgen angehängt werden als Dokumentation des Entwicklungsprozesses. Eine automatisierte Protokollierung der Änderungen findet jedoch nicht statt und es stehen insgesamt keine spezifischen Mittel zur Verfügung, die die Bildung von Konsens zwischen den Kodierern fördert.

- Es fehlt eine Möglichkeit, mit der Ontologie und einer Dokumentsammlung gemeinsam zu interagieren, wodurch zugleich eine wichtige Quelle von Feedback für den Ontologieentwickler entfällt.

- Es stehen keine automatisierten Evaluationsverfahren zur Verfügung, die die Güte der Ontologie messen, etwa im Sinne der Abdeckung von Daten einer repräsentativen Ressource oder von signifikativen Textbausteinen in einer Textsammlung.

- Die Maintenance und Adaption der Ressource erfolgt mit denselben technischen Mitteln wie in der Kodierungsphase. In dieser Hinsicht ist eine spezifische technische Unterstützung wünschenswert, die die speziellen Aufgaben in diesen Phasen berücksichtigt, insbesondere wenn Texte als Datenquelle herangezogen werden.

2.3 Ziele

Wie im letzten Abschnitt dargestellt, können im Fall der beim Aufbau des CoGE-Netzes verfolgten Methodologie verschiedene Vorschläge zur Verbesserung des gesamten Lebenszyklus' der Ontologie gemacht werden, insbesondere aus technischer Perspektive. In diesem Abschnitt erfolgt eine Einschränkung auf konkrete Ziele. Methodologisch betrachtet konzentrieren sich diese Ziele auf zwei besonders aufwändige Schritte, nämlich die *Population* und die fortlaufende Maintenance eines EFGT-Netzes, sowie einen weiteren Eckpunkt: Eine Endanwendung, die als Kontrollschritt im Lebenszyklus der Ontologie dient.

Ziel 1: *Technische Unterstützung für die halbautomatische Integration von (semi-)strukturierten Daten.* Die Ontologie wird bisher durch die händische Einarbeitung einzelner Einträge aufgebaut. Diese Vorgehensweise ist geeignet, um den Kern eines Wissensbereich zu kodieren, aber auf Grund der Anzahl der Einträge extrem ineffizient beim systematischen Ausbau (*Population*) der Ressource. Neben der potentiell sehr großen Anzahl der Entitäten, die für bestimmte Klassen, wie z.B. *Personen*, vorzunehmen sind, muss der Aufwand berücksichtigt werden, der bei der Erfassung und Spezifikation der linguistischen Repräsentation der einzelnen Konzepte anfallen kann. Dabei sind die Daten, die dem Kodierer als Ergebnis der Wissensakquisiton vorliegen, meistens vorstrukturiert, insbesondere dann, wenn sie aus bereits existierenden Ressourcen stammen oder das Ergebnis automatisierter Extraktionsverfahren sind. Aus diesem Grund wird angestrebt, die halbautomatische Integration (semi-)strukturierter Daten technisch zu unterstützen. Mit "halbautomatisch" ist gemeint, dass aus den eingehenden Daten Einträge in der Wissensressource automatisch generiert werden sollen, jedoch die Ontologieentwickler die Möglichkeit erhalten, diesen Prozess zu überwachen und bei Bedarf einzugreifen. Insbesondere soll die Integration von Teil-Hierarchien und Daten, die facettierte Einträge darstellen, ermöglicht werden. Insgesamt soll dadurch ein effizienter Aufbau der Ressource erreicht werden, ohne dabei die Qualität der Einträge zu gefährden.

Ziel 2: *Bereitstellung einer Endanwendnug, die Bezug auf Textsammlungen nimmt.* Endanwendungen können als Kontrolle des Entwicklungsprozesses der Ontologie bzw. als Katalysator fungieren, die die Entwicklung eines neuen thematischen Bereichs auslösen. Die zu Beginn dieser Arbeit vorhandene Endanwendung, das EFGT-Netz-Browser-Plug-In, trägt dazu bei, einen Eindruck über die Ressource anhand einzelner Webdokumente zu gewinnen. Das Plug-In nimmt jedoch keinen Bezug auf eine repräsentative Dokumentensammlung, so dass die Nützlichkeit der Ressource in ihrer Gesamtheit bei typischen Einsatzszenarien, wie beispielsweise der thematischen Navigation oder semantischen Suche nicht überprüft werden kann. Aus diesem Grund wird als Ziel festgelegt, eine Anwendung bereitzustellen, die ein Feedback über die Ressource beim Zugriff auf Dokumentensammlung vermittelt, so dass die kodierte thematische Einordnung von Entitäten überprüfbar und ein Rahmen für die menschliche Interaktion mit dem EFGT-Netz geboten wird.

Ziel 3: *Dokumentenzentrierte Wissensakquisition und Maintenance innerhalb der Anwendung.* In vielen thematischen Bereichen stellen Textdokumente oft die einzige Quelle für die Wissensakquisition dar, wie bei aktuellen Ereignissen, aktuellen Themen aus dem Pressebereich, usw.. Texte bringen die Aktualität ein, die für die fortlaufende Plege der Ressource nötig ist. Die im zweiten Ziel angestrebte Endanwendung soll in einem nächsten Schritt so ausgebaut werden, dass dem Benutzer im Zuge der Interaktion mit der Anwendung möglich ist, in einem Dokument vorkommende neue Konzepte in die Ressource leicht zu übernehmen. Im Rahmen dieser Anwendung, die ja einen Eindruck vom Entwicklungsstand der Wissensressource in Bezug auf eine Dokumentsammlung vermitteln soll, sollen sich somit neue Einträge akquirieren lassen, so dass das jeweilige EFGT-Netz fortlaufend gepflegt oder ggf. an eine bestimmte Dokumentensammlung adaptiert werden kann.

Zusammenfassend soll durch die Umsetzung dieser Ziele der Aufbau und die fortlaufende Pflege von EFGT-Netzen effizienter gestaltet werden. Hierfür wird technisch betrachtet von der beschriebenen Situation ausgegangen, in der die grundlegende Infrastruktur zur Speicherung und zum Aufbau von EFGT-Netzen sowie Schnittstellen zur Anbindung externer Applikationen bereits existieren. Durch die Festlegung auf obige Ziele entstehen neue Anforderungen an die Technik, beispielsweise:

- Möglichkeiten zur Definition und Ausführung von Schemata, mit denen ausgehend von bestehenden Daten Einträge im EFGT-Netz erzeugt werden können (Ziel 1).
- Abgleichsmechanismen, um generierte Einträge mit bereits bestehenden zu vergleichen und dadurch zu verhindern, dass doppelte Einträge entstehen, Konflikte erkennen, usw. (Ziel 1).
- Verfahren, die eine thematische Einordnung von Texten vornehmen (Ziel 2).
- Methoden zur Identifikation von sprachlichen Ausdrücken in Texten, die Kandidaten für neue Konzepte im EFGT-Netz darstellen (Ziel 3).

Zusätzliche Anforderungen, die sich bei der Umsetzung der festgelegten Ziele stellen, werden im weiteren Verlauf der Arbeit vorgestellt.

2.3.1 Ausblick

In den folgenden Kapiteln werden die in diesem Abschnitt definierten Ziele ausgearbeitet. Mit dem in Kap. 3 vorgestellten Upload-Tool wird auf das Ziel der Integration semistrukturierter Daten eingegangen. Kap. 4 beschreibt eine Endanwendung zur Suche und Navigation in Dokumentenarchiven, in der ein EFGT-Netz eingesetzt wird. Die Erweiterung dieser Anwendung zur textbasierten Wissensakquisition und Maintenance erfolgt schließlich in Kap. 5.

Kapitel 3

Integration von semi-strukturierten Daten: Das *Upload-Tool*

Das erste in Kapitel 2 definierte Ziel, das zum effizienten Aufbau von EFGT-Netzen führen soll, ist eine technische Infrastruktur hervorzubringen, mit der die semiautomatische Integration von Daten in das EFGT-Netz möglich ist. Durch die automatische Vorgenerierung von Einträgen auf der Grundlage vorhandener, in der Phase der Wissensakquisition gewonnener Daten soll dem Ontologieentwickler ein wesentlicher Teil der manuellen Arbeit abgenommen werden. Gleichzeitig soll der Ontologieentwickler in der Lage sein, den Integrationsprozess zu überwachen und zu steuern, sodass insgesamt der semiautomatische, datengesteuerte Aufbau der Ressource möglich wird.

In diesem Kapitel wird auf dieses Ziel näher eingegangen. Im ersten Teil wird die Grundidee vorgestellt, die der Generierung von Einträgen ausgehend von vorhandenen Daten zugrunde liegt, sowie Anforderungen an ein System erfasst, das diese Grundidee umsetzen und den semiautomatischen Aufbau eines EFGT-Netzes ermöglichen soll. Im zweiten Teil des Kapitels wird eine abstrakte Lösung zur Realisierung der Anwendungslogik eines solchen Systems vorgestellt: Eine spezielle Sprache, mit Hilfe derer sich die Integration von Daten in das EFGT-Netz durchführen lässt. Der dritte Teil des Kapitels widmet sich dem konkreten System, dem sogenannten *Upload-Tool*, das im Rahmen dieser Arbeit implementiert wurde und mit dem auf die speziellen Anforderungen eines menschlichen Benutzers eingegangen wird, sowie dessen Einsatz beim Ontologieaufbau.

3.1 Ausgangspunkt: Muster bei der Kodierung von EFGT-Netz-Einträgen

Nachdem der Kern entwickelt, die wichtigsten thematischen Bereiche und die geographischen und temporalen Achsen eines EFGT-Netzes ausgebaut worden sind, werden in der Phase der *Population* eines EFGT-Netzes vorrangig Einträge vorgenommen, die einzelne benannte Entitäten darstellen und in der Struktur des Netzes Blätter bilden (s. Abschnitt *Kodierung*, S. 50). Diese benannten Entitäten können als einzelne Instanzen verschiedener

Entitätenklassen wie *Personen*, *Gemeinden Deutschlands*, *Baufirmen in Großbritannien*, usw. aufgefasst werden, die innerhalb einer bestimmten Klasse ähnlich konzeptualisiert werden und charakteristische, binäre Relationen zu anderen Konzepten des Netzes eingehen. So kann jede einzelne Instanz des Konzepts *Personen* als eine Entität mit einer bestimmten Nationalität, Beruf, Geburtsjahr und einer starken Beziehung zu einem bestimmten Thema aufgefasst werden. Bei der Kodierung dieser Instanz-Konzepte spiegeln sich die charakteristischen Relationen im Identifikator des Eintrags als *Facetten* wider. Beispielsweise könnte man den ID-String für das Konzept *Gabriel García Márquez* folgendermaßen spezifizieren:

(e([*Kolumbien*]&[*Schriftsteller*]&[*Jahr 1928*]&[*Fantastischer Realismus*]).26)

wobei hier die einzelnen Komponenten des ID-Strings durch ihre natürlichen Namen im EFGT-Netz ersetzt wurden, um die Lesbarkeit und die Kodierung charakteristischer Relationen zu anderen Konzepten im Netz zu verdeutlichen. Einen validen ID-String erhält man einfach dadurch, dass für jede Angabe der Form [*Konzept*] der korrespondierende ID-String dieses *Konzepts* ersetzt wird. So müsste man im obigen Ausdruck die Komponente [*Jahr 1928*] durch ihren Identifikator (t((T(t.2).2)&(t.11)).59.1.3.9) ersetzen. Die charakteristischen Relationen des Konzepts *Gabriel García Márquez* spiegeln sich in der Kodierung als &-Summe von Werten charakteristischer Facetten, wobei der Eintrag mit dem Typ e und einem noch nicht belegten, "frischen" Index (im Bsp. 26) lokal eingeführt wird. Führt man für jede Facette im obigen Ausdruck eine Variable ein, die je nach *Person* einen konkreten Wert annimmt, kann man den ID-String aller Instanzen der Klasse *Personen* mit folgendem "Muster" zusammenfassen:

(e(*Land* & *Beruf* & *Geburtsjahr* &*Thema*).*n*)

wobei Variablen *kursiv* geschrieben sind und *n* einen frischen Index für jede Person bezeichnet. Ähnliche Muster lassen sich für weitere semantische Klassen wie *Gemeinden Deutschlands*, *Weißweine*, usw. angeben, bei denen alle Elemente innerhalb der Klasse "analytisch" ähnlich konzeptualisiert werden.

Auch wenn ein Muster für die Kodierung der Instanzen einer Klasse angegeben werden kann, ist die Kodierung einer solchen Klasse von benannten Entitäten mit signifikantem Aufwand verbunden: Für jeden einzelnen Eintrag müssen die Facetten mit konkreten Werten belegt werden, wofür einschlägiges Wissen vorausgesetzt wird und die passenden Konzepte im bereits bestehenden EFGT-Netz gefunden werden müssen. So wird insgesamt für die Kodierung einer ganzen Klasse sehr viel kontingentes Wissen vorausgesetzt, da im Prinzip jede einzelne Facette sehr viele Werte annehmen kann und sich insgesamt eine hohe Multiplizität bei der Kombination mehrerer Facetten ergeben kann. Dem kann nur begegnet werden, indem eine gezielte, automatisierte Wissensakquisition durchgeführt oder auf geeignete externe Wissensressourcen zurückgegriffen wird.

Für das Ziel, den anvisierten Wissensbereich angemessen abzudecken und die Aktualität der Ontologie zu wahren, hat die *Population* der Ontologie und die regelmäßige Pflege dieser potenziell sehr großen, im Prinzip offenen Klassen von benannten Entitäten oder Lexikon-Klassen jedoch eine zentrale Bedeutung, da benannte Entitäten als Verankerung

des EFGT-Netzes in Texten dienen und somit den Ausgangspunkt für das thematische *Reasoning* darstellen (s. Kapitel 2). Die Existenz von Kodierungsmustern öffnet in dieser Hinsicht eine Tür zur Automatisierung der *Population* der Ontologie mit Lexikon-Klassen: Das Muster kann als Schablone gedacht werden, die für jede neu einzuführende Entität mit akquirierten Daten, die als Werte für die Facetten fungieren, ausgefüllt wird und zu einem neuen Eintrag in der Ressource führt. Hier gilt es, vorhandene, vorstrukturierte Daten in Form von Einträgen auf das EFGT-Netz abzubilden.

Muster in der Kodierung von Einträgen sind auch in anderen Bereichen zu beobachten. So können beispielsweise bei der Kodierung einer Taxonomie im EFGT-Netz die einzelnen *ist-eine-Art-von*-Kanten, die jedes Element der Taxonomie mit dessen Elternknoten verbinden, als Facetten aufgefasst werden. Ein Muster für den ID-String aller Elemente der Taxonomie, die drei Elternteile haben, könnte also folgendermaßen aussehen:

(e(*Elternteil1* & *Elternteil2* & *Elternteil3*) . *n*)

Hier gilt, die richtigen Elternteile für jedes Element der Taxonomie einzusetzen.

Wie an den Beispielen zu sehen ist, ist das jeweilige Muster je nach betrachteter Klasse oder Bereich semantisch anders zu interpretieren. Die Beziehung des neuen Eintrags zu den einzelnen Konzepten, die im ID-String-Muster die Variablen belegen, kann semantisch betrachtet eben eine Rolle, eine *ist-eine-Art-von*-, d.h. taxonomische Beziehung, oder auch andere Relationen darstellen. In dem Sinne, in dem die unterschiedlichen, für jeden Bereich relevanten Beziehungen durch deren Kodierung im ID-String auf die Vorfahre-Nachfahre-Relation des Netzes abgebildet werden, ist diese strukturelle Relation *semantisch überladen*.

Eine andere Art von Mustern kann bei der linguistischen Repräsentation des Konzeptes und der Definition anderer Attribute beobachtet werden. So ist denkbar, bei der Klasse der *Gemeinden Deutschlands* neben dem Namen der Gemeinde im Nominativ die Genitiv-Form durch Hinzufügung eines *s* automatisch im Eintrag zu erzeugen oder aus den akquirierten Daten die Anzahl der Anwohner als Attribut zu übernehmen. Bei der Erarbeitung eines Systems zur automatischen Generierung von Einträgen ist ebenfalls diese Art von Mustern zu berücksichtigen, da sie einen relevanten Teil des Aufwands der Kodierung ausmachen.

Ausgehend von der Hauptidee, Kodierungsmuster als Grundlage zu nehmen, werden in diesem Abschnitt zunächst weitere Anforderungen an ein System zur semiautomatischen Integration von Daten identifiziert. Im weiteren Verlauf des Kapitels erfolgt dann die Entwicklung technischer Lösungen für diese Anforderungen. An dieser Stelle sei darauf hingewiesen, dass eine Betrachtung des Problems der Wissensakquisition in dieser Arbeit ausgeklammert wird. Wie bereits erwähnt, wird hier die Annahme gemacht, dass auf geeignete bestehende Ressourcen zurückgegriffen oder bekannte Techniken der Informationsextraktion, wie der Einsatz von Wrappertools und Crawler, Analyse mit regulären Ausdrücken usw., eingesetzt werden können, um nötige Daten für den Aufbau der Ontologie bereitzustellen.

3.2 Anforderungen an ein System zur Datenintegration

An ein System, das ausgehend von der oben beschriebenen Grundidee die semiautomatische Integration von (semi-)strukturierten Daten in das EFGT-Netz ermöglichen soll, werden zum einen Anforderungen an die interne Logik des Systems gestellt, zum anderen solche, die aus der Perspektive des Ontologieentwicklers maßgeblich sind.

Die Erfordernisse an die interne Anwendungslogik des Systems zum Abbilden von Daten und automatisierten Erzeugen von Einträgen lassen sich näher beschreiben:

- *Definition von Schemata für Einträge*, insbesondere für den ID-String und andere Attribute des Eintrags. Desweiteren ist ein Mechanismus erforderlich, mit dem in einem Schema auf bereits im EFGT-Netz bestehenden Einträge, auf die die zu generierenden Einträge aufbauen sollen, Bezug genommen werden kann.

- *Mechanismen zur Belegung der Schemata mit Daten und Generierung von Einträgen.* Hierbei besteht das Hauptproblem darin, unterschiedliche Formate der Daten zu berücksichtigen und generische Mechanismen anzubieten, mit denen auf Regionen dieser Daten zugegriffen werden kann, sodass Schemata sukzessiv zu Einträgen vervollständigt werden können. Da eine weitgehende Trennung vom Wissensakquisitionsprozess angestrebt wird, bevorzugt man hierfür generische Modelle, die wenige Voraussetzungen an die Daten stellen. Insbesondere sollen sowohl strukturierte als auch semistrukturierte Daten berücksichtigt werden.

- *Alinierungs- und Abgleichsmechanismen*, die generierte und im EFGT-Netz bestehende Einträge miteinander vergleichen und mögliche Konflikte und Duplikate identifzieren.

Zusammengenommen lassen sich diese drei Ziele mit der Entwicklung einer *Integrationssprache* gleichsetzen, mit Hilfe derer sich Transformationen vorhandener Daten in Einträge der Ressource spezifizieren lassen.

Aus dem Betrachtungswinkel des Ontologieentwicklers stellen sich eine Reihe von Anforderungen, die dem Zweck einer Professionalisierung der Entwicklungsumgebung dienen und eine effiziente Steuerung und Überwachung des Integrationsprozesses großer Datenmengen ermöglichen sollen:

- Es sollen sich Schemata angeben und editieren lassen, mit denen die Integration von Daten im Sinne der oben besprochenen Integrationssprache durchgeführt wird.

- Um die Qualität der Ressource nicht zu gefährden, müssen die generierten Einträge sowie die Ergebnisse des Abgleichs mit dem EFGT-Netz dem Ontologieentwickler präsentiert werden, der darüber entscheidet, welche Einträge tatsächlich übernommen werden. In dieser Hinsicht ist der Ansatz halbautomatisch. Da große Datenmengen verarbeitet werden sollen, ist eine geeignete Visualisierung von auftretenden Konflikten und Problemen nötig, um die "optische" Überwachung des Integrationsprozesses zu ermöglichen.

- Die *Population* der Ontologie mit dem System soll ein sicherer, versuchender Vorgang sein, in dem Sinne, dass verschiedene Schemata hintereinander ausprobiert, Einträge vorgeneriert und getestet werden können, ohne dass dabei die Gefahr besteht, inkorrekte Einträge in der Ressource zu erzeugen. Werden Einträge vom Benutzer bestätigt und dem EFGT-Netz hinzugefügt, so sollen diese protokolliert werden und die Möglichkeit bestehen, sie rückgängig zu machen. Außerdem sollen gängige Operationen für die Verwaltung von Daten wie etwa die Verwendung von Ergebnissen früherer Alinierungen sowie das "Einfrieren" des Integrationsprozesses bestimmter Daten zur Fortsetzung zu einem späteren Zeitpunkt, usw. unterstützt werden.

- Es wird eine webbasierte Lösung angestrebt, die die gleichzeitige Arbeit mehrerer Ontologieentwickler am EFGT-Netz ermöglicht.

- Das System soll für naive Benutzer verwendbar sein für den Fall, dass das Kodierungsmuster vorgegeben ist. D.h., die Bedienung des Systems setzt weniger Wissen über den EFGT-Formalismus voraus als die Kodierung einzelner Einträge von Grund auf, sodass Hilfs-Ontologieentwickler eingesetzt werden können, die sich auf die Überprüfung der generierten Einträge konzentrieren.

- Bei Auftreten von Konflikten und gängigen Problemen, die zur Generierung fehlerhafter Einträge führt, möchte man diese Probleme *on-the-fly* lösen können, d.h. ohne die Kodierungsoberfläche aufzurufen, die Anwendung zu verlassen und den Integrationsprozess unterbrechen zu müssen. Auch dies soll die Verwendung des Systems durch naive Anwender unterstützen.

- Die Multilingualität soll dadurch unterstützt werden, dass in Schemata Eintragsattribute in mehreren Sprachen gleichzeitig definiert werden können, sodass ggf. auch bestehende Einträge erweitert werden.

3.3 Eine Sprache zur Integration von Daten in EFGT-Netze

Nachdem die Anforderungen an ein System zur Datenintegration näher beschrieben wurden, wird an dieser Stelle eine Lösung für den Teil der Anforderungen entwickelt, die sich auf die Logik eines solchen Systems beziehen und unter dem Stichwort *Integrationssprache* zusammengefasst wurden. Diese Lösung, mit der neue Einträge aus vorhandenen Daten generiert und in ein EFGT-Netz integriert werden sollen, wird in den nächsten Abschnitten zunächst von einer abstrakten, von einer bestimmten Implementierung unabhängigen Perspektive aus beschrieben. Mit dem Einsatz dieser Integrationssprache im Rahmen eines konkreten Systems befasst sich der dritte Teil des Kapitels im Abschnitt 3.4.

3.3.1 Schemata zur Definition generischer EFGT-Netz-Einträge

Wie in der Einleitung beschrieben, will man Muster bei der Kodierung einer Klasse von ähnlich konzeptualisierten benannten Entitäten in einem generischen Eintrag zusammenfassen, der für jede einzelne Entität der Klasse mit passenden Daten ausgefüllt wird und zu einem tatsächlichen Eintrag im EFGT-Netz führt. Insbesondere muss in einem solchen generischen Eintrag – im folgenden auch *Template* oder *EFGT-Netz-Eintragsschema* genannt – ein Muster für den ID-String angegeben werden, in dem Variablen Platzhalter für existierende Konzepte im EFGT-Netz darstellen, zu denen jede einzelne Entität für die Klasse charakteristische Beziehungen eingeht. Außerdem muss in einem solchen Eintragsschema die Repräsentation des Konzepts durch linguistische und andere Angaben spezifiziert werden können.

Für die Spezifikation von solchen Templates wird in diesem Abschnitt eine Sprache entwickelt, die bezüglich der Angabe generischer Identifikatoren eine natürliche Erweiterung der Sprache der ID-Strings darstellt (s. Abschnitt 2.1.2, S. 37).

Die Template-Sprache umfasst einen *konstruktiven Teil* zur Angabe neuer Einträge und einen *variablen Teil*, der Templates generisch macht. Auf den variablen Teil wird später eingegangen, es sei zunächst der konstruktive Teil betrachtet.

Ein neuer Eintrag in der Wissensressource wird in der Template-Sprache durch eine Deklaration der Form

erweiterter ID-String
Liste von Attributen

spezifiziert. Der *erweiterte ID-String* in dieser Deklaration stellt eine Konstruktionsvorschrift für den ID-String des neuen Eintrags dar, die Bezug auf bereits im Netz existierende Konzepte nimmt. Die Liste der Attribute gibt die linguistische Repräsentation und die optionalen Attribute des Eintrags vor. Die Liste der Attribute wird dem korrespondierenden erweiterten ID-String explizit zugewiesen, indem die äußere Klammer des erweiterten ID-Strings mit einem sogenannten *Anker* markiert wird und dieser gefolgt von einem Punkt vor jeder Attributdeklaration der Liste verwendet wird:

(`NeuesKonzept` *Rest des erweit. ID-Strings*)

`NeuesKonzept`.AttributDeklaration1
`NeuesKonzept`.AttributDeklaration2
...

Ein Anker ist schlicht ein alphanumerischer Name, in obiger Deklaration `NeuesKonzept`. Die Syntax der Attributdeklarationen ist in Abbildung 3.1 zusammengefasst. Als Beispiel definiert folgende Deklaration eine einfache linguistische Repräsentation in einem neuen Eintrag für das Konzept *Landkreise in Hessen*:

(`NeuesKonzept` *Rest des erweit. ID-Strings*)

```
NeuesKonzept.name.de.nomen = "Landkreise in Hessen"
NeuesKonzept.variante.de.nomen = "Landkreise Hessens"
```

In dem Beispiel wird dem neuen Konzept ein Namen (Annotationsrolle `name`) auf Deutsch (Sprache `de`) mit dem linguistischen Typ `nomen` sowie eine Variante mit dem Wert `Landkreise Hessens` zugewiesen.

$$
\begin{array}{lcl}
\mathbb{A}\text{ttr}\mathbb{L}\text{ist} & ::= & (\mathbb{A}\text{nchor} \,.\, \mathbb{A}\text{ttr}\mathbb{D}\text{ecl})^{+} \\
\mathbb{A}\text{ttr}\mathbb{D}\text{ecl} & ::= & \mathbb{L}\text{ing}\mathbb{A}\text{ttr} \mid \mathbb{O}\text{pt}\mathbb{A}\text{ttr} \\
 & & \\
\mathbb{L}\text{ing}\mathbb{A}\text{ttr} & ::= & \mathbb{A}\text{nno}\mathbb{R}\text{olle1} \,.\, \mathbb{L}\text{ang} \,.\, \mathbb{L}\text{ing}\mathbb{T}\text{ype} \texttt{ = "} \mathbb{V}\text{alue} \texttt{"} \\
\mathbb{A}\text{nno}\mathbb{R}\text{olle1} & ::= & \texttt{name} \mid \texttt{variante} \mid \ldots \\
\mathbb{L}\text{ang} & ::= & \texttt{de} \mid \texttt{en} \mid \ldots \\
\mathbb{L}\text{ing}\mathbb{T}\text{ype} & ::= & \texttt{nomen} \mid \texttt{eigenname} \mid \texttt{adj} \mid \ldots \\
 & & \\
\mathbb{O}\text{pt}\mathbb{A}\text{ttr} & ::= & \mathbb{A}\text{nno}\mathbb{R}\text{olle2} \,.\, \mathbb{D}\text{ata}\mathbb{T}\text{ype} \texttt{ = "} \mathbb{V}\text{alue} \texttt{"} \\
\mathbb{A}\text{nno}\mathbb{R}\text{olle2} & ::= & \texttt{biolink} \mid \texttt{gebjahr} \mid \texttt{homepage} \mid \ldots \\
\mathbb{D}\text{ata}\mathbb{T}\text{ype} & ::= & \texttt{string} \mid \texttt{integer} \mid \texttt{url} \mid \ldots \\
 & & \\
\mathbb{A}\text{nchor} & ::= & \text{Alphanum}^{+} \\
\mathbb{V}\text{alue} & ::= & \text{String}
\end{array}
$$

Abbildung 3.1: Grammatik zur Spezifikation von Attributlisten. Eine Liste von Attributdeklarationen (Kategorie $\mathbb{A}\text{ttr}\mathbb{L}\text{ist}$) besteht aus beliebig vielen Ausdrücken der Kategorie $\mathbb{A}\text{ttr}\mathbb{L}\text{ist}$, die durch einen Anker ($\mathbb{A}\text{nchor}$) qualifiziert sind. Damit können sowohl die linguistische Repräsentation (Deklarationen mit $\mathbb{L}\text{ing}\mathbb{A}\text{ttr}$) des Konzepts als auch die optionalen Attribute ($\mathbb{O}\text{pt}\mathbb{A}\text{ttr}$-Deklarationen) definiert werden. Für eine Erklärung der verschiedenen Annotationsrollen und deren möglichen Typen vgl. die Beschreibung in Kap. 2, S. 40.

Um das Eintragsschema zu vervollständigen, muss ein erweiterter ID-String angegeben werden, von dem im obigen Beispiel die äußere Klammerung mit dem Anker für die Attributliste sichtbar ist. Mit diesem erweiterten ID-String setzt man den Eintrag in Beziehung zu bestehenden Konzepten, um die Relationen der Konzeptualisierung auszudrücken. Hierfür können die Identifikatoren der angesprochenen Erweiterung der Sprache der ID-Strings verwendet werden, die $\mathbb{ID}\mathbb{S}\text{tring}^{gen}$ genannt wird, in Anlehnung an den Namen der ID-Strings in Kap. 2, $\mathbb{ID}\mathbb{S}\text{tring}$ (s. Abb. 2.1, S. 38).

Neben der Verwendung der Anker-Klammerung besteht die Erweiterung hauptsächlich in der Einführung eines verweisenden Sprachkonstrukts, sogenannter *Query-Komponenten*. Eine Query-Komponente dient dazu, mittels eines sprachlichen Ausdrucks auf ein im EFGT-Netz existierendes Konzept zuzugreifen und dessen ID-String für die Konstruktion des Identifikators im Eintragsschema zu verwenden. Syntaktisch wird eine Query-Komponente durch Angabe des sprachlichen Ausdrucks zwischen eckigen Klammern angegeben. Beispielsweise stellt `[Hessen]` einen Verweis auf das Konzept *Hessen* dar. Schließlich erfolgt die Spezifikation des ID-Strings unter Bezugnahme bestehender Einträge durch die Kombination von Query-Komponenten mit den beiden syntaktischen Operationen der ID-String-Syntax, der lokalen Einführung und der `&`-Summenbildung. Wird im obigen Beispiel

das Konzept *Landkreise in Hessen* als eine Menge von geographischen Bereichen konzeptualisiert (Typ `G`), die unter dem zugehörigen Bundesland lokal einzuführen ist, so lässt sich das entsprechende Eintragsschema folgendermaßen angeben:

$$(\,_{\texttt{NeuesKonzept}}\,\texttt{G[Hessen].n})$$

```
NeuesKonzept.name.de.nomen = "Landkreise in Hessen"
NeuesKonzept.variante.de.nomen = "Landkreise Hessens"
```

Die Semantik dieses Templates ist ein neuer Eintrag in der Wissensressource. Angenommen, der ID-String für das Konzept *Hessen* ist `(g(G.2).7)` im EFGT-Netz, würde dieses Template den neuen ID-String `(G(g(G.2).7).1)` für *Landkreise in Hessen* generieren. Die abgewandelte Form der lokalen Einführung mit dem generischen Index `.n` sorgt dafür, dass ein frischer Index für den neuen ID-String ermittelt wird. Im Beispiel wird also zusätzlich angenommen, dass unter *Hessen* keine weitere lokale Einführung des Typs `G` vorhanden ist, sodass `n = 1` einen frischen Index darstellt. Der neue Eintrag kann formal als ein Paar $\varepsilon = (\phi, \Omega)$ aufgefasst werden, wobei ϕ den erfolgreich konstruierten ID-String darstellt und Ω die Liste der Attribute, die ϕ über den Anker zugeordnet wurden:

$$\begin{aligned}\varepsilon = \;& (\,\texttt{(G(g(G.2).7).1)}\,,\\ & \{\,(\texttt{name.de.nomen}\,,\text{"Landkreise in Hessen"}),\\ & \;(\texttt{variante.de.nomen}\,,\text{"Landkreise Hessens"})\,\}\,)\end{aligned}$$

Attribute werden ebenfalls als Paare aufgefasst, wobei das erste Element des Paars der komplexe Typ, der durch die Kombination der Annotationsrolle, Sprache, usw. entsteht, und das zweite Element der Wert des Attributs ist.

$$\begin{aligned}\mathbb{IDS}\text{tring}^{gen} \;&::=\; (\,_{\mathbb{A}\text{nchor}}\,\mathbb{T}\text{yp}\;\mathbb{IDS}\text{tring}'\;\texttt{.n)} \mid (\,_{\mathbb{A}\text{nchor}}\,\mathbb{IDS}\text{tring}'\;\texttt{\&}\;\mathbb{IDS}\text{tring}'\,\texttt{)}\\ \mathbb{IDS}\text{tring}' \;&::=\; \texttt{( )} \mid \texttt{[}\,\mathbb{V}\text{alue}\,\texttt{]} \mid \texttt{(}\,\mathbb{T}\text{yp}\;\mathbb{IDS}\text{tring}^{gen}\;\texttt{.}\;\mathbb{I}\text{ndex}\,\texttt{)} \mid\\ &\qquad \texttt{(}\,\mathbb{IDS}\text{tring}^{gen}\;\texttt{\&}\;\mathbb{IDS}\text{tring}^{gen}\,\texttt{)}\end{aligned}$$

Abbildung 3.2: Syntax der erweiterten Identifikatoren $\mathbb{IDS}\text{tring}^{gen}$ (vgl. Abb. 2.1)

Analog zum Beispiel können Query-Komponenten als Summanden einer `&`-Summe verwendet werden. Die Syntax der erweiterten ID-Strings $\mathbb{IDS}\text{tring}^{gen}$ ist in Abb. 3.2 zusammengefasst. Wie aus dieser Grammatik ersichtlich, können mehrere erweiterte Identifikatoren ineinander verschachtelt werden, sodass das Eintragsschema auf kompakte Weise mehrere Einträge definieren kann. Die durch die Anker markierten Teile des Identifikators entsprechen jeweils einem neu generierten ID-String, dem im Eintragsschema mittels des korrespondierenden Anchors Attributdeklarationen zugewiesen werden. Somit kann das Beispiel-Template erweitert werden, um einen bestimmten Landkreis in die Menge der *Landkreise Hessens* einzuführen:

```
( Kreis g( Kreismenge G[Hessen].n).n)

Kreismenge.name.de.nomen      = "Landkreise in Hessen"
Kreismenge.variante.de.nomen  = "Landkreise Hessens"
Kreis.name.de.nomen           = "Main-Taunus-Kreis"
```

Ein Template ist valide, wenn allen im erweiterten Identifikator vorkommenden Anker jeweils mindestens eine Attributdeklaration zugewiesen wird.

Der eigentliche Nutzen der Templates entsteht jedoch erst durch die Verwendung von Variablen, sodass ein Eintragsschema generisch wird und durch die wiederholte Belegung der Variablen mit Werten aus akquirierten Daten insgesamt alle Einträge für die anvisierte Klasse von benannten Entitäten erzeugt werden kann. Variablen können entweder in einer Query-Komponente für den generischen Verweis auf bestehende Konzepte eingesetzt werden oder in den Werten der Attributzuweisungen. Letzteres kann erreicht werden, indem die Regel für $\mathbb{V}$alue aus Abb. 3.1 modifiziert wird:

$$\mathbb{V}\text{alue} \quad ::= \quad \text{String} \mid \text{String?}\ (\ \mathbb{V}\text{ar String?}\ )^{+}$$

Nach der Erweiterung der Regel um den Ausdruck auf der rechten Seite der Alternative können Variablen (dargestellt durch $\mathbb{V}$ar) in beliebige Strings eingebettet werden. Wie der Zugriff auf Daten über Variablen erfolgen kann und die Ausdrücke in $\mathbb{V}$ar aussehen, wird in Abschnitt 3.3.3 behandelt: Je nach Art der Daten werden dort unterschiedliche Arten von Variablen definiert, um je nach Format auf verschiedene Regionen der Daten zugreifen zu können und eine sinnvolle Belegung des Eintragsschemas zu ermöglichen. Um die Mächtigkeit von Variablen an dieser Stelle dennoch zu vermitteln, sei ein weiteres Beispiel betrachtet: Das Template in Abb. 3.3 – im folgenden *Bezirkstemplate* genannt – enthält kursiv geschriebene *Variablen* und kodiert Einträge für Mengen von Bezirken verschiedener schweizer Kantone, ist im Aufbau jedoch analog zum Template für die *Landkreise Hessens*. Die Variable *Kanton* steht für den Kanton. In die zugehörige Bezirksmenge werden einzelne Bezirke eingeführt, deren deutscher Namen durch die Variable *Bezirk* repräsentiert wird. Das frühere Beispiel mit den *Landkreisen Hessens* erweiternd, wird hier außerdem noch ein Eintrag für die Hauptstadt des Bezirks (Variable *Hauptstadt*) generiert, die im erweiterten ID-String durch die Summenbildung des Bezirks mit dem Konzept *Hauptstädte* modelliert wird.

```
( Hauptstadt ( Bezirk g( Bezirksmenge G[ Kanton].n).n)&[Haupstädte])

Bezirksmenge.name.de.nomen      = "Bezirke in Kanton"
Bezirksmenge.variante.de.nomen  = "Bezirke Kantons"
Bezirk.name.de.nomen            = "Bezirk"
Hauptstadt.name.de.nomen        = "Hauptstadt"
```

Abbildung 3.3: Eintragsschema zur Kodierung von Bezirken und deren Hauptstädte für schweizerische Kantone (sog. *Bezirkstemplate*)

Kanton	Bezirk	Hauptstadt
Thurgau	Bezirk Weinfelden	Weinfelden
Thurgau	Bezirk Bischofszell	Bischofszell
Wallis	Bezirk Brig	Brig-Glis
Wallis	Bezirk Visp/Viège	Visp
Wallis	Bezirk Östlich-Raron/Rarogne oriental	Mörel

Tabelle 3.1: Geographische Daten bezüglich der Schweiz

Sind beispielsweise als Ergebnis der Wissensakquisition die Daten in Tabelle 3.1 vorhanden und werden die Variablen des Templates Zeile für Zeile der Tabelle entsprechend den Namensspalten mit Werten belegt, so werden für jede Zeile drei neue Einträge generiert. Abbildung 3.4 stellt die erzeugten Einträge für die erste Zeile der Tabelle dar.

```
{  ( (G(g(G().2).3).1), { (name.de.nomen, "Bezirke in Thurgau"),
       (variante.de.nomen, "Bezirke Thurgaus") } ),
   ( (g(G(g(G().2).3).1).3), { (name.de.nomen, "Bezirk Weinfelden") } ) ,
   ( ((g(g(G(g(G().2).3).3).1).3)&(G((G().1)&(F().2)).3)),
       { (name.de.nomen, "Weinfelden")} )    }
```

Abbildung 3.4: Mit dem *Bezirkstemplate* generierte Einträge für die erste Zeile in Tab. 3.1

Semantik von EFGT-Netz-Eintragsschemata

Im vorherigen Abschnitt wurde während der Beschreibung der Syntax der EFGT-Netz-Eintragsschemata erwähnt, dass die Semantik eines solchen Schemas nach Belegung der Variablen mit Werten und dessen Auswertung eine Menge von Einträgen ist. Während die Vervollständigung der Attributdeklarationen im Template mit Werten aus den Daten – soweit diese Werte ermittelbar sind – stets zu korrekten Attributzuweisungen im jeweiligen Eintrag führt, liefert die Auswertung des erweiterten Identifikators nicht in jedem Fall einen definierten Wert für die verschiedenen ID-Strings.

Bei der Konstruktion von ID-Strings für die Einträge wird der erweiterte Identifikator von innen nach außen und von links nach rechts entsprechend der mit Ankern markierten Klammerung ausgewertet. Beispielsweise bedeutet dies für das Bezirkstemplate, dass zunächst versucht wird, den ID-String für den Anker `Bezirksmenge` aufzubauen, anschließend für `Bezirk` und schließlich nach der Auswertung der Query-Komponente `[Hauptstädte]` den mit `Hauptstadt` markierten ID-String. Der Grund dafür, dass die Auswertung des erweiterten ID-Strings fehlschlagen kann, sind eingebettete Query-Komponenten, die unter Umständen nicht durch einen definierten ID-String ersetzt werden können. Ein ID-String zur Ersetzung einer Query-Komponente wird ermittelt, indem im EFGT-Netz nach einem Eintrag gesucht wird, der die Angabe in der Query – nachdem Variablen belegt worden sind – als Attribut hat. Diese Suche entspricht einer Funktion $getEntries$: String $\longrightarrow \mathcal{P}(E)$, wobei $\mathcal{P}(E)$ die Potenzmenge aller Einträge im EFGT-

Netz darstellt. Liefert diese Funktion für eine Query-Angabe σ $getEntries(\sigma) = \{\varepsilon\}$ mit $\varepsilon \in E$, so wird die Query-Komponente bei der Auswertung des erweiterten Identifikators des Templates durch den ID-String von ε ersetzt. In den anderen Fällen, $getEntries(\sigma) = \emptyset$ und $getEntries(\sigma) = A$ mit $|A| > 1$, ist der Wert für die Query-Komponente undefiniert und die Auswertung des Templates schlägt fehl.

Bei der Berechnung von $getEntries(\sigma)$ kann nützlich sein, σ mit den in der Wissensressource definierten Attributen auf *linguistische Ähnlichkeit* hin zu testen, statt lediglich auf Gleichheit. Die Nützlichkeit dieses Vergleichs beruht darauf, die Suche permissiver zu machen und dadurch die Wahrscheinlichkeit, dass die Auswertung einer Query-Komponente fehlschlägt, geringer zu halten. Der Vergleich auf linguistische Ähnlichkeit hin kann ebenfalls als eine Funktion aufgefasst werden:

$$lingSim : \text{String} \longrightarrow \mathcal{P}(\text{Attr})$$

mit Attr der Menge aller im EFGT-Netz definierten Attribute. Mit Hilfe dieser Funktion kann $getEntries$ folgendermaßen definiert werden:

$$getEntries(\sigma) = \{\varepsilon \in E | \varepsilon \text{ hat ein Attribut aus } lingSim(\sigma)\}$$

Für die Praxis gilt, eine linguistische Ähnlichkeitsfunktion zu finden, die gegenüber den in Daten vorkommenden Varianten tolerant ist, ohne dass jedoch $getEntries$ unnötig viele Ergebnisse liefert. In der Literatur sind eine Reihe von linguistischen Ähnlichkeitsfunktionen vorgeschlagen worden, die sich je nach linguistischer Klasse der Angabe (Mehrwortlexem, Eigenname, einfaches Wort, usw.) und Sprache unterschiedlich eignen. Für einen Überblick wird auf Euzenat und Shvaiko (Euzenat and Shvaiko, 2007, S. 74 ff) verwiesen.

Ein weiterer dynamischer Aspekt bei der Auswertung des in einem Template angegebenen erweiterten ID-Strings ist die Bestimmung eines Wertes für den generischen Index in lokalen Einführungen. Nach Abb. 3.2 kommen generische Indizes in Ausdrücken der Form $(_{\mathbb{A}\text{nchor}}\mathbb{T}\text{yp}\mathbb{ID}\mathbb{S}\text{tring}'\ \texttt{.n})$ vor. Bei der Auswertung von ID-Strings dieser Form können verschiedene Fälle unterschieden werden, wobei nur die betrachtet werden, bei denen der eingebettete ID-String (oben durch $\mathbb{ID}\mathbb{S}\text{tring}'$ dargestellt) bereits erfolgreich aufgebaut werden konnte. Ist $(_\rho\tau\phi\ \texttt{.n})$ ein solcher Ausdruck, so wird zunächst der ID-String $(\tau\phi\ \texttt{.333})$ normalisiert (s. S. 39). Ist der normalisierte String $(\tau\phi'\ \texttt{.333})$, so werden anschließend alle ID-Strings ermittelt, die den Teilstring $(\tau\phi'\ \texttt{.}$ enthalten. Folgende Fälle ergeben sich dann:

1. Ist im EFGT-Netz kein ID-String mit dem Teilstring $(\tau\phi'\ \texttt{.}$ vorhanden, so erhält der mit ρ markierte Eintrag den Identifikator $(\tau\phi'\ \texttt{.1})$, d.h. $\texttt{n} = \texttt{1}$.

2. Lassen sich im EFGT-Netz ein oder mehrere Einträge mit einem ID-String finden, der $(\tau\phi'\ \texttt{.}$ als Teilstring enthält, so wird der maximale Index max ermittelt, der in allen ermittelten ID-Strings dem Teilstring folgt. Anschließend wird $\texttt{n}$ folgendermaßen bestimmt:

(a) Findet sich in der Datenbank ein Eintrag mit einem gesamten Identifikator der Form $(\tau\phi' . x)$, wobei $1 \leq x \leq max$ und sind die Attribute dieses Eintrags *unifizierbar* (s.u.) mit den Attributen des im Template mit ρ markierten Eintrags, so werden beide Einträge gleichgesetzt, d.h. in diesem Fall $\mathtt{n} = x$.

(b) Ansonsten $\mathtt{n} = max + 1$.

Dass die Attribute zweier Einträge *unifizierbar* sind, bedeutet, dass alle Attribute eines Eintrags im anderen enthalten sind. Zwei Attribute sind gleich, wenn sie komponentenweise verglichen gleich sind (komplexer Typ und Wert).

Durch den im Fall 2.(a) nötigen Vergleich zwischen Einträgen ist die Auswertung eines Templates mit der Alinierung der generierten Einträge zu den Einträgen im EFGT-Netz eng verzahnt. Den Details dieser Alinierung widmet sich der nächste Abschnitt.

3.3.2 Abgleich und Alinierung generierter Einträge

Das Ziel der Alinierung der mit einem Eintragsschema erzeugten Einträgen mit dem EFGT-Netz ist die Ermittlung von Kandidaten, die neue Einträge darstellen und in die Ontologie übernommen werden können.

Zur Alinierung der generierten Einträge mit dem EFGT-Netz kann der Vergleich einerseits basierend auf dem ID-String (*logisches Matching*) erfolgen und anderseits auf Grundlage der definierten Attribute, insbesondere der linguistischen Repräsentation (*linguistisches Matching*). Für einen einzigen erzeugten Eintrag reicht bei der Alinierung aus, ihn mit den Einträgen der Ontologie zu vergleichen. Da der Zweck eines Templates die Erzeugung einer Vielzahl von Einträgen ist, die erst nach Bestätigung durch den Ontologieentwickler in das EFGT-Netz übernommen werden, werden die erzeugten, noch nicht bestätigten Einträge solange in einem *Eintragsspeicher* gehalten. Im Normalfall muss also der Abgleich eines neu erzeugten Eintrags mit den bereits vorhandenen Einträgen sowohl im Eintragsspeicher als auch in der Wissensressource durchgeführt werden. Somit umfasst der *erweiterte Eintragsraum* alle Einträge, die sich im Speicher oder im EFGT-Netz befinden, und alle im folgenden beschriebenen Vergleichsoperationen werden, wenn nichts anderes explizit vermerkt ist, in diesem Raum durchgeführt.

Prozedural beginnt der Abgleich eines generierten Eintrags mit dem Aufbau des im Template spezifizierten ID-Strings für den Eintrag. Wie im Abschnitt über die Semantik eines Schemas dargestellt, ist dieser Aufbau im Falle einer lokalen Einführung unter Umständen mit einem linguistischen Matching (Test auf Unifizierbarkeit) zur Ermittlung eines passenden Indizes verbunden. Ansonsten kann das logische Vorhandensein des Eintrags einfach durch einen Test entschieden werden, ob der ID-String im erweiterten Eintragsraum bereits definiert ist.

Nach dem logischen Matching wird das linguistische Matching durchgeführt: Es wird nach Einträgen gesucht, die mit dem generierten Eintrag unifizierbar sind (Definition s. Abschnitt 3.3.1), wobei im Gegensatz zum logischen Abgleich hier das Ergebnis mehr als einen Eintrag enthalten kann. Ist das Ergebnis des linguistischen Matchings nicht leer, so ist der generierte Eintrag bereits *linguistisch vorhanden*.

Ling. Vorhanden	**Part. Ling. Match**	**Logisch Vorhanden**	**Part. Log. Match**	**Name**	**Code**	**Interpretation: *Aktion***
	-		-	Pot. neuer Eintrag	00	Neuer Eintrag: *übernehmen*
x	-		-	Namenskonflikt	10	1) Homonyme Konzepte: *Vorzugsbez. ändern/ Eintrag unverändert übernehmen* 2) Unterschiedliche Modellierung desselben Konzepts: *ID-String ändern/ auswählen*
x	-	x	-	Konzeptmatch	11	Vorhandener Eintrag: -
	-	x	-	Logischer Konflikt	01	1) Komplementäre ling. Repräsentation: *Attributrepr. vereinigen* 2) Sem. verschiedene Konzepte: *ID-String beider Einträge verfeinern*
	x	x		pot. Konzeptmatch (ling. Überlappung)	21	Fall 01 Interpr. 1: *Attributrepräsentation vereinigen*
x			x	pot. Konzeptmatch (log. Überlappung)	12	Fall 10 Interpr. 2: *ID-String ändern/ auswählen*
	x		x	pot. Konzeptmatch (ähnliche Konzepte)	22	1) Konzeptidentität: *ID-Strings und linguistische Repräsentation vereinigen* 2) Verschiedene Konzepte: *keine*
	x			ling. Warnung	20	Potenzieller Namenskonflikt: *ggf. wie Fall 10*
			x	log. Warnung	02	Potenzieller logischer Konflikt: *ggf. wie Fall 01*

Tabelle 3.2: Interpretation der Fälle beim Abgleich generierter Einträge

Aus der Kombination der Testergebnisse auf logisches und linguistisches Vorhandensein ergeben sich die vier ersten Fälle der Tabelle 3.2. Ein generierter Eintrag wird als *neuer Eintrag* interpretiert, wenn weder auf der logischen noch auf der linguistischen Ebene ein Konflikt vorhanden ist. In diesem Fall kann der Eintrag einfach in die Wissensressource übernommen werden. Ergibt sich ein *Namenskonflikt* (zweiter Fall), hat das zwei mögliche Interpretationen: Eine besteht darin, dass die beiden Einträge, wie von ihren unterschiedlichen ID-Strings ausgedrückt, semantisch verschiedene Konzepte darstellen, aber *homonym*, d.h. in ihrem Bezeichner ununterscheidbar sind. Durch die Übernahme des neuen Eintrags würde eine Ambiguität in der Wissensressource entstehen. Um dies zu vermeiden, kann etwa die Vorzugsbezeichnung des generierten Eintrags geändert werden. Die andere Interpretation eines Nameskonflikts fasst sowohl den generierten als auch den linguistisch gleich repräsentierten, bestehenden Eintrag als dasselbe abstrakte Konzept auf, wobei dieses aber in den beiden Einträgen logisch unterschiedlich modelliert wird. In diesem Fall muss entschieden werden, welcher ID-String übernommen werden soll. Insbesondere ist es interessant zu prüfen, ob einer der beiden ID-Strings eine Verfeinerung des anderen darstellt. Ein *logischer Konflikt* (dritter Fall) kann ebenfalls zweifach ausgelegt werden. Eine Möglichkeit ist, dass die linguistischen Repräsentationen der beiden Einträge komplementär sind und etwa ein Synonym der anderen erfasst. In diesem Fall liegt die Vereinigung beider Attributmengen nahe. In der anderen, konversen Interpretation bezeichnen beide linguistisch gleiche Einträge unterschiedliche semantische Konzepte, deren logische Modellierung zu grob ist, um auf der Ebene der ID-Strings unterschieden werden zu können. Schließlich liegt ein *Konzeptmatch* vor, wenn im erweiterten Eintragsraum sowohl linguistisch als auch logisch der generierte Eintrag von einem vorhandenen ununterscheidbar ist.

Zusätzliche Informationen durch partielles Matching

Mit der Alinierung generierter Einträge will man verhindern, dass *semantisch* redundante Einträge in der Datenbank entstehen. Der im vorherigen Abschnitt beschriebene Abgleich identifiziert jedoch potenziell redundante Einträge auf Grund *syntaktischer* Hinweise, der logischen und linguistischen Repräsentation. Bei der Interpretation der Konfliktfälle muss der Ontologieentwickler selbst entscheiden, ob der Eintrag semantisch redundat ist oder nicht, und ihn entsprechend verwerfen oder verändern. Zusätzliche Information, die die Interpretation der unklaren Fälle unterstützen kann, lässt sich gewinnen, indem die Abgleichmethode permissiver gemacht und dadurch bei der Alinierung Einträge berücksichtigt werden, die sich in der Attributrepräsentation oder im ID-String ähneln. Mit Hilfe dieser zusätzlichen Information ist zu erwarten, dass sich der Integrationsprozess beschleunigen lässt und sicherer bezüglich der Vermeidung redundanter Einträge wird.

Beim Vergleich der Attributrepräsentation zweier Einträge lässt sich das in Abschnitt 3.3.1 beschriebene, exakte Verfahren auf unterschiedliche Weise durchlässiger machen. Anstatt die vollkommene Unifizierbarkeit der beiden Attributrepräsentationen zu betrachten, kann etwa gefordert werden, dass nur sie einen bestimmten Anteil gemeinsamer Attribute haben. Eine andere Möglichkeit ist, Attribute als gleich gelten zu lassen, wenn sie ungeachtet des Typs den gleichen Wert haben. Außerdem kann der Vergleich der Attributwerte

mit Hilfe der Funktion *lingSim* approximativ gemacht werden: Demnach ist ein Attribut (τ, σ) – wobei wie bisher τ den komplexen Typ aus der Kombination der Annotationsrolle mit zusätzlichen Typen und σ den Wert darstellt – mit einem Attribut ρ eines anderen Eintrags gleichzusetzen, wenn $\rho \in lingSim(\sigma)$. In der Implementierung (s. Abschnitt 3.4) ist ein Eintrag ε' aus dem erweiterten Eintragsraum ein *partielles linguistisches Match* des generierten Eintrags ε_{gen}, wenn sich ein Attribut $\rho \in \varepsilon'$ und ein Attribut $\phi = (\tau, \sigma) \in \varepsilon_{gen}$ finden lassen, so dass $\rho \in lingSim(\sigma)$.

Die Definition einer approximativen Vergleichsvorschrift, die die semantische Nähe zweier ID-Strings bewertet, ist durchaus schwieriger. Bei einem solchen approximativen logischen Matching kann zum einen die Gestalt der beiden ID-Strings miteinander verglichen werden. Hierbei müssen unterschiedliche Faktoren und Fälle betrachtet werden. Beispielsweise kann man die semantische Nähe zweier Identifikatoren, die beide eine `&`-Summe darstellen und mehrere Komponenten gemeinsam haben, unterschiedlich bewerten, je nachdem, wie groß der Anteil der nicht geteilten Komponenten ist. Ein weiterer Fall könnte der sein, in dem ID-Strings der Form

$$S_{gen} = \texttt{(e(}\, X \,\texttt{\&}\, Y \,\texttt{\&}\, Z \,\texttt{).1)}$$

$$S_{found} = \texttt{(e(}\, A \,\texttt{\&}\, Y' \,\texttt{\&}\, Z \,\texttt{).3)}$$

gegeben sind. Hier stehen X, Y, usw. für spezifischen Komponenten und Y' für einen Vorfahren von Y in einem bestimmten EFGT-Netz. Zur Bewertung der semantischen Nähe dieser beiden ID-Strings können im Prinzip unterschiedliche Kriterien wie die Anzahl der gemeinsamen Komponenten, die Übereinstimmung der Typen der beiden ID-Strings, der Grad der semantischen Verwandtheit zwischen Y und Y', usw. herangezogen werden. Bei der Betrachtung dieser und weiterer Fälle ist es unklar, welche Kriterien ein gutes Maß der semantischen Nähe auf der Grundlage des Aufbaus der Identifikatoren a priori liefern. Zum anderen scheint es sinnvoll zu sein, beim approximativen logischen Matching die Topologie des Netzes zu berücksichtigen, in denen die zu vergleichenden ID-Strings eingebettet sind. Im obigen Beispiel kann die semantische Nähe von S_{gen} und S_{found} davon abhängen, wieviele Konzepte im spezifischen Netz Y und Y' voneinander trennen. Bei der Betrachtung von ausschließlich zwei ID-Strings kann nichts darüber ausgesagt werden, welche Struktur beide Konzepte im Netz verbindet. Je nach Größe des betrachteten EFGT-Netzes und je nachdem, welche ID-Strings als Konzepte umgesetzt sind, kann semantische Nähe topologisch betrachtet unterschiedlich verstanden werden.

Gute Kriterien für Maße semantischer Nähe scheinen sich demnach nur experimentell bestimmen zu lassen. Außerdem scheint die Interpretation eines solchen Maßes für den Benutzer schwer zu sein, wenn sich ohnehin keine guten Vergleichskriterien a priori angeben lassen. Ein *partielles logisches Matching* wird bis zur Entwicklung einer geeigneten Vorschrift nur als theoretische Möglichkeit in Tab. 3.2 aufgenommen. In der Implementierung liegt ein partielles logisches Match vor, wenn zwei ID-Strings syntaktisch die gleiche lokale Einführung darstellen und sich nur im Index unterscheiden.

Tabelle 3.2 erfasst die zusätzlichen Fälle, die sich bei der Alinierung generierter Einträge ergeben, wenn Ergebnisse eines partiellen linguistischen bzw. logischen Abgleichverfahrens

als Zusatzinformation dienen. Die ersten zwei Fälle unterstützen die Interpretation des Abgleichs im Konfliktfall. Wenn beispielsweise der ID-String des generierten Eintrags bereits vorhanden ist und zusätzlich eine partielle Übereinstimmung der Attributrepräsentation mit demselben Eintrag vorliegt (Fallcode 21), spricht das für die erste Interpretation des vorhandenen logischen Konflikts (Fallcode 01). Die Fälle mit dem Code 22, 20 und 02 stellen Zusatzinformationen im Fall eines potenziellen neuen Eintrags dar.

Auf die in der Tabelle vorgeschlagenen Aktionen zur Lösung der Konflikte wird in der Implementierung (s. Abschnitt 3.4) eingegangen.

3.3.3 Datenformate und Instanziierung von Schemata

Bei der Beschreibung der Sprache für die EFGT-Netz-Eintragsschemata in Abschnitt 3.3.1 wurde darauf hingewiesen, dass ein Schema erst durch die Verwendung von Variablen generisch wird, sodass durch mehrfache Instanziierung dieser Variablen immer wieder neue Einträge entstehen. Der Mechanismus, mit dem diese Variablen mit Daten belegt werden, wurde dabei noch offen gehalten. Darauf wird nun näher eingegangen.

Ist ein Template mit Variablen vorgegeben, lassen sich im Allgemeinen aus einer bestimmten Datenquelle mehrere Variablenbelegungen dafür gewinnen. Zunächst muss geklärt werden, was sinnvolle Variablenbelegungen sind. Außerdem muss bei der Erzeugung von Variablenbelegungen das spezifische Format der Daten berücksichtigt werden. Im folgenden wird ein Kriterium aufgestellt, mit dem sich sinnvolle Variablenbelegungen identifizieren lassen. Anschließend wird der Einsatz dieses Kriteriums bei unterschiedlichen Datenformaten beschrieben. Im einzelnen werden tabellarische Daten, Daten, die als typisierte Felder vorliegen und schließlich XML-Daten betrachtet. Somit wird die prozedurale Bedeutung eines Templates, wenn spezifischen Daten in einem dieser Formate vorliegen, beleuchtet.

Funktionale Abhängigkeiten als Orientierung

Bei der Entwicklung von Mechanismen zur Definition von Variablenbelegungen ist folgende Beobachtung nützlich: Bei der Kodierung eines Konzeptes hängt die Wahl der Facetten in der Konzeptualisierung vom jeweiligen zu kodierenden Konzept ab. So bedingt das Konzept *Gabriel García Márquez* die Wahl von *Kolumbien* als in Relation stehendem Land bei der Kodierung. In einem Template drückt sich diese Abhängigkeit dadurch aus, dass die Variablen, die von einer Klammerung mit einem bestimmten Anker umfasst werden, in Abhängigkeit vom Konzept belegt werden, für das der aktuelle Anker steht. Beispielsweise sollte im Template auf Abb. 3.3 (s. S. 68) für die Variable *Kanton* der Wert `Thurgau` eingesetzt werden, wenn der Anker `Bezirk` über die Variable *Bezirk* mit `Bezirk Weinfelden` belegt wird und dieses Konzept repräsentiert. Anders ausgedrückt, liegen die vom Anker abhängigen Variablen *im syntaktischen Skopus des Ankers*. Diese Abhängigkeit ist *funktional* in dem Sinne, dass das Konzept die Belegung der Variablen eindeutig festlegt. Eine funktionale Abhängigkeit besteht außerdem noch bei verschachtelten Templates zwischen den im erweiterten ID-String vorkommenden Ankern: Liegt ein Anker A im syntaktischen

Skopus eines anderen Ankers B, so hängt die Belegung vom Konzept A von der des Konzepts B ab, da die Kodierung von A auf der Kodierung von B aufbaut. So bedingt im Bezirkstemplate (Abb. 3.3) die Belegung der *Hauptstadt* mit `Weinfelden` den Wert `Bezirk Weinfelden` für *Bezirk*.

Wie unten für die drei unterschiedlichen Formate näher erläutert wird, kodieren unterschiedliche Datenformate auf verschiedene Weise implizit funktionale Abhängigkeiten. Der Leitgedanke bei der Definition von Mechanismen zur Schema-Instanziierung besteht darin, zu ermöglichen, dass in der Datenquelle mittels des Formats kodierte funktionale Abhängigkeiten auf solche im Template abgebildet werden können. Demnach müssen für jedes einzelne Format spezifische Verweismechanismen entwickelt werden, sodass auf im Format kodierte Abhängigkeiten zugegriffen werden kann.

Das Prinzip, sich funktionaler Abhängigkeiten bei der Definition von Variablenbelegungen zu bedienen, schränkt die Datenquellen ein, die mit der hier entwickelten Schemasprache verarbeitet werden können: Das Format der Datenquelle muss dieselben funktionalen Abhängigkeiten kodieren, die im Template eine Rolle spielen. Anders ausgedrückt, muss das Format der Datenquelle die Relationen in der Ontologie auf *natürliche Weise* kodieren. In der Praxis stellte sich heraus, dass das keine wirkliche Einschränkung darstellt. Der Vorteil dieses Ansatzes liegt darin, dass die Daten im Originalformat belassen werden können und nicht erst transformiert werden müssen, um die funktionalen Abhängigkeiten im Template widerzuspiegeln.

Tabellarische Daten

In einer Tabelle lässt sich üblicherweise mindestens eine Spalte finden, die als Schlüssel für die Werte in den anderen Spalten dient. D.h., dass die Werte in einer Zeile vom Wert der Schlüsselspalte funktional abhängen. Beispielsweise in Tab. 3.1 (s. S. 69) bedingen sowohl die Spalte *Bezirk* als auch die Spalte *Hauptstadt* jeweils die anderen zwei Spalten funktional, sodass Ketten von funktionalen Abhängigkeiten entstehen. Aus jeder Zeile kann deswegen ein Tupel extrahiert werden, mit dem die im Bezirkstemplate von Abb.3.3 kodierten Abhängigkeiten instanziiert werden können.

Die im Abschnitt 3.3.1 noch kursiv geschriebenen *Variablen* werden in der Syntax der Templatesprache mit dem Symbol `#` markiert. Sind die Spalten der Tabelle benannt, so kann im Template nach diesem Symbol der Name der Spalte angegeben werden. Ansonsten werden die Spalten intern von links nach rechts nummeriert, sodass der Verweis über `#1`, `#2`, `#3`, usw. erfolgt. Das Template wird dann zeilenweise ausgewertet. Da für alle Formate die Variablen mit `#` anfangen, wird die Information über das Format der Daten – in diesem Fall "Tabelle" – dem Template weitergegeben, um die korrekte Interpretation der Variablen sicherzustellen. Abbildung 3.5 zeigt das Bezirkstemplate mit konkreten Variablen, die auf die Daten in Tab. 3.1 verweisen. Bei den in einem String eingebetteten Variablen dient die Klammerung `{...}` dazu, die Variable vom umgebenden String zu trennen.

Die Mindestanforderung der Integrationssprache an Tabellen ist das Vorhandensein einer Schlüsselspalte. Tabellen eignen sich daher besonders als Format für Templates mit einem einzigen Anker und die Integration von Klassen facettierter Konzepte, die unabhängig

```
( Hauptstadt ( Bezirk g( Bezirksmenge G[ #Kanton ].n).n)&[Hauptstädte])

  Bezirksmenge.name.de.nomen      = "Bezirke in #Kanton"
  Bezirksmenge.variante.de.nomen  = "Bezirke #{Kanton}s"
  Bezirk.name.de.nomen            = "#Bezirk"
  Hauptstadt.name.de.nomen        = "#Hauptstadt"
```

Abbildung 3.5: Das *Bezirkstemplate* mit Verweisen auf die Spalten von Tabelle 3.1

voneinander sind, wie Personen, Firmen, usw. Wie am Bsp. gezeigt, lassen sich aber auch hierarchische Daten wie die geographischen Daten aus Tab. 3.1 in einer Tabelle kodieren, es gibt allerdings hierfür kompaktere Datenformate, wie etwa XML (s. S. 81 ff).

Bei Tabellen ist es oft zweckmäßig, bestimmte Zeilen zu überspringen oder in Abhängigkeit der Belegung einer bestimmten Spalte verschiedene erweiterte ID-Strings auszuwerten. Hierfür stellt die Templatesprache eine bedingte Klausel (`if-then-else`) zur Verfügung, mit der Bedingungen an Variablen gestellt werden können. Die konkrete Syntax ist im Anhang A angegeben.

Typisierte Felder

Mit *typisierten Feldern* wird ein Datenformat bezeichnet, in dem Daten zeilenweise strukturiert sind, wobei jede Zeile aus einer variablen Anzahl von Feldern der Form *Typ*: *Wert* besteht. Die übliche Interpretation dieses Formats besteht darin, dass ein fester Typ in jeder Zeile ein Feld markiert, das als Schlüssel für die anderen Felder dient, d.h. die anderen Werte in der Zeile hängen funktional von der Angabe in diesem Feld ab, wobei diese Abhängigkeit typisiert ist. Dieses Format bietet mehr Flexibilität für die Kodierung von Daten in der Hinsicht, dass die Anzahl der Felder in jeder Zeile im Gegensatz zu den Spalten in einer Tabelle unterschiedlich sein kann und die Felder in unterschiedlicher Abfolge der Typen vorkommen kann. Typisierte Felder sind in der Computerlinguistik ein übliches Format, das beispielsweise zur Kodierung von Lexika verwendet wird, die aus Einträgen bestehen, für die eine unterschiedliche Anzahl von typisierten Merkmalen oder eine unterschiedliche Anzahl von Übersetzungen in verschiedenen Sprachen angegeben sind, usw. Dieses Format eignet sich durch seine Flexibilität und Einfachheit als Basisformat, in das aus komplex strukturierten Daten extrahierte Relationen transformiert werden können. Insbesondere können in diesem Format Hierarchien kodiert werden, bei denen ein Element mehrfache ummittelbar übergeordnete Elternelemente haben kann.

Die geographischen Daten über die Schweiz sind in Abb. 3.6 als typisierte Felder kodiert. Zur Definition einer Variablenbelegung für das Template werden die Typen der Felder in den Variablen angegeben. Für die Variable, die den Namen des Konzepts bereitstellt, das durch den äußersten Anker aufgebaut wird, setzt man den Typ des Schlüssels ein. Abb. 3.7 zeigt das Bezirkstemplate für den Einsatz mit den typisierten Feldern aus Abb. 3.6. Als Schlüssel dienen die Felder vom Typ *hpt*, die eine Eins-zu-eins-Beziehung mit dem *bzk*-Feld

```
bzk:Weinfelden; hpt:Weinfelden; kt:Thurgau;
bzk:Bischofszell; hpt:Bischofszell; kt:Thurgau;
bzk:Brig; hpt:Brig-Glis;kt:Wallis;
bzk:Visp; fr:Viège; hpt:Visp; kt:Wallis;
bzk:Östlich-Raron; fr:Rarogne oriental; kt:Wallis; hpt:Mörel;
```

Abbildung 3.6: Geographische Daten bezüglich der Schweiz als typisierte Felder. Bedeutung der Typen: *bzk*: Bezirk; *fr*: französischer Name des Bezirks; *kt*: Kanton; *hpt*: Hauptstadt;

```
( Hauptstadt ( Bezirk g( Bezirksmenge G[#kt].n).n)&[Hauptstädte])

 Bezirksmenge.name.de.nomen      = "Bezirke in #kt"
 Bezirksmenge.variante.de.nomen  = "Bezirke #{kt}s"
 Bezirk.name.de.nomen            = "#bzk"
 Hauptstadt.name.de.nomen        = "#hpt"
 Bezirk.variante.fr.nomen        =? "#{fr}s"
```

Abbildung 3.7: Das *Bezirkstemplate* mit Verweisen auf die typisierten Felder in Abb. 3.6

haben, das in der Formatierung in Abb. 3.6 als Schlüssel suggeriert wird.

Die letzte Zeile des Templates in Abb. 3.7 dient dazu, den optionalen französischen Namen, der im eventuell vorkommenden Feld vom Typ *fr* angegeben wird, als Attribut für das generierte Konzept zu erfassen. Optionale Attribute werden in der Templatesprache mit dem Symbol `=?` gesondert ausgezeichnet, da in der Regel nicht belegte Variablen zum Fehlschlagen der Auswertung des Templates führen.

Ein weiteres Merkmal des Datenformats typisierte Felder ist, dass in einer Zeile mehrere Felder vom selben Typ angegeben sein können. Um diesen Fall zu behandeln, wird für dieses Format eine spezielle Art von Variablen ("Multivariablen") zur Verfügung gestellt, die mit dem Symbol `#*` gekennzeichnet werden. Eine solche Variable kann in der Templatesprache in einer Attributdeklaration oder in einer Query-Komponente verwendet werden und führt in der Auswertung zu einer Vervielfältigung des Elements, in das sie eingebettet ist. Folgendes Beispiel erläutert diesen Effekt. Das Template auf Abb. 3.8 macht Gebrauch von Multivariablen und dient zur Integration der Daten in Abb. 3.9, die eine Taxonomie von Krankheiten darstellen. Jede Zeile in Abb. 3.9 kodiert ein Element der Taxonomie, wobei der deutsche Name im Feld vom Typ *mainDE* angegeben ist sowie weitere optionale Namen auf Englisch (*en*) oder Latein (*lt*). Die unmittelbaren Oberbegriffe jeden Elements in der Taxonomie sind mit der unterschiedlich langen Liste von Feldern des Typs *kat* kodiert. Im Template (Abb. 3.8) wird über die Variable `#mainDE` der deutsche Name für das Konzept angegeben. Optionale Attribute dienen dazu, die englischen und lateinischen Bezeichnungen zu erfassen, wobei der Verweis auf die entsprechenden Felder über die Multivariablen `#*en` und `#*lt` erfolgt. Die Semantik einer solchen Spezifikation ist diese: Es werden so viele englische (lateinische) Varianten als Attribut gespeichert wie möglich, eine

```
( Krankheit F&*[#*kat].n)

Krankheit.name.de.nomen       = "#mainDE"
Krankheit.variante.en.nomen   =? "#*en"
Krankheit.variante.lt.nomen   =? "#*lt"
```

Abbildung 3.8: Ein Template zur Integration der Taxonomie von Krankheiten in Abb. 3.9

Variable `#en` (`#lt`) unterschiedlich zu belegen. Das Attribut wird den Daten entsprechend nach Bedarf vervielfältigt. Die Verwendung von Multivariablen in Attributdeklarationen ist nur bei optionalen Attributen möglich.

Im erweiterten String des Schemas für die Krankheitstaxonomie wird ebenfalls die Multivariable `#*kat` eingesetzt als Verweis auf die unterschiedliche Anzahl von Oberbegriffen in jeder Zeile der Taxonomie. Diese Multivariable ist in ein Konstrukt eingebettet, das die `&`-Summe einer unterschiedlichen Anzahl von Query-Komponenten darstellt. Bei der Auswertung wird für diese Summe pro Feld des Typs *kat* in einer Zeile eine Query-Komponente bereitgestellt. So wird das Konstrukt

```
&*[#*kat]
```

– eine *variable Query-Summe* – für die erste Zeile in Abb. 3.9 folgendermaßen belegt und ausgewertet:

```
[Krankheit]&[Immunologie]&[Allergologie]
```

Diese Summe von Query-Komponenten stellt den Kern für die lokale Einführung von Typ `F` dar, als die die Krankheit schließlich kodiert wird.

Die Verwendung von variablen Query-Summen im erweiterten ID-String stellt eine Erweiterung der Syntax von $\mathbb{ID}\mathbb{S}\text{tring}^{gen}$ dar. In Abb. 3.2 (S. 67) muss dementsprechend die Regel für $\mathbb{ID}\mathbb{S}\text{tring}'$ um eine weitere Alternative für die Syntax der variablen Query-Summen ergänzt werden (s. Anhang A).

Kodierung von Hierarchien mittels typisierter Felder. Die Kodierung von Hierarchien stellt besondere Anforderungen an das Format typisierte Felder. Neben dem Vorhandensein eines Schlüsselfelds und Feldern für die Eltern des Schlüsselfelds in der Hierarchie müssen beispielsweise die Zeilen in einer bestimmten Reihenfolge angegeben werden: Bei der Auswertung einer bestimmten Zeile (einer Kategorie in der Hierarchie) muss sichergestellt sein, dass alle die Oberkategorien, auf die in der Zeile verwiesen wird, entweder im EFGT-Netz vorhanden oder durch Auswertung einer entsprechenden, vorausgehenden Zeile entstanden sind. Der im Laufe der Auswertung eines Templates aufgebaute Eintragsspeicher (s. Abschnitt *Abgleich und Alinierung generierter Einträge*, S. 71) unterstützt diese Anforderung, indem er ermöglicht, bei der Auswertung einer bestimmten Zeile auf Konzepte zu verweisen, die im Rahmen der Evaluation derselben Datei bereits generiert worden sind.

```
mainDE:Infektionskrankheiten;en:infectious disease;en:infections;kat:Mikrobiologie;\
                                                          kat:Krankheiten nach Typ;
mainDE:Parasitäre Infektion;de:Parasiteninfektion;en:parasitic infections;kat:Parasitologie;\
                                                          kat:Infektionskrankheiten;
mainDE:Wurminfektionen;en:worm infections;en:worm infestations;kat:Parasitäre \
                                                          Krankheiten;
mainDE:Milbeninfektionen;en:mite infections;kat:Parasitäre Krankheiten;
mainDE:Infektionen durch Protozoen;de:Protozoeninfektion;en:protozoan infections;\
                                                          kat:Parasitäre Krankheiten;
mainDE:Giardiasis;lt:Lambliasis;en:giardiasis;en:lambliasis;kat:Protozoeninfektion;\
                                                          kat:Infektionskrankheiten;
...
```

Abbildung 3.9: Eine Taxonomie von Krankheiten im Format typisierter Felder

Im einzelnen sind bei der Kodierung von Hierarchien mittels typisierter Felder folgende Eigenschaften der Datei sicherzustellen:

- Die Felder in der Datei müssen syntaktisch wohlgeformt und die Verwendung der Typen für den Schlüssel und die Oberkategorien muss konsistent sein.
- Jede Zeile muss frei von redundanten Oberbegriff-Angaben sein. Dies sind solche, die sich aus der Transitivität der Oberbegriff-Relation in der kodierten Hierarchie ableiten lassen. So ist in Abb. 3.10 *Infektionskrankheiten* ein Oberbegriff von *Protozoeninfektion*. Die Angabe von *Infektionskrankheiten* als Oberbegriff für den Eintrag *Giardiasis* mit dem unmittelbaren Oberbegriff *Protozoeninfektion* in Abb. 3.9 ist redundant, weil sich diese Beziehung aus der Struktur der Hierarchie ableiten lässt.
- Die Kategorien, auf denen die gesamte Hierarchie aufbaut und im EFGT-Netz vorausgesetzt werden, müssen mit denen übereinstimmen, die sich aus der Kodierung der Hierarchie ableiten lassen. Dies bedeutet, es müssen alle Kategorien in der Datei gefunden werden, die dort erwähnt sind und nicht durch andere Kategorien definiert werden. Diese Liste muss mit den Konzepten im EFGT-Netz verglichen werden.
- Die in der Datei kodierte Hierarchiestruktur muss frei von Zyklen sein, d.h. zirkuläre Kategoriedefinitionen sind nicht erlaubt.
- Die Sortierung der Zeilen muss gewährleisten, dass die Kategorien, auf die in einer Zeile als Oberbergiff verwiesen wird, bereits in einer früheren Zeile aufgebaut oder im EFGT-Netz definiert worden sind.

Da die händische Überprüfung dieser Eigenschaften insbesondere für große Hierarchien kaum möglich ist, wurde ein Hilfsprogramm entwickelt, welches diese Eigenschaften überprüft. Dieses Programm dient bei der händischen Kodierung und der automatischen Extraktion von Hierarchien als Kontrolle, bevor die Integration der Hierarchie mittels eines Eintragsschemas versucht wird.

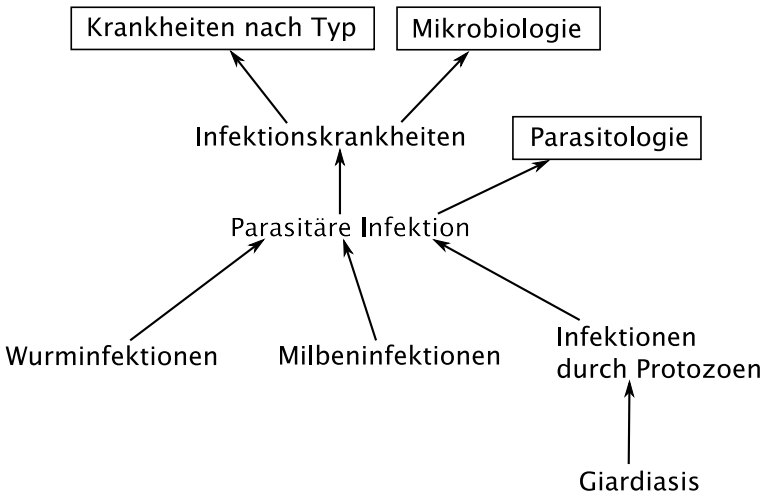


Abbildung 3.10: Struktur der in Abb. 3.9 kodierten Krankheitstaxonomie. Die eingerahmten Begriffe werden im EFGT-Netz vorausgesetzt.

XML-Daten

XML eignet sich besonders für die Kodierung hierarchischer Daten, die eine baumartige Struktur darstellen. Ein XML-Dokument kann als Baum mit beschrifteten Knoten aufgefasst werden. Insbesondere unterscheidet das *Document Object Model* (vgl. DOM, 2005), ein Modell für den sprach- und plattformunabhängigen Zugriff auf HTML- und XML-Dokumente, drei Arten von Knoten: Elementknoten, Attributknoten und Textknoten. Auf einem solchen Baum lassen sich strukturelle Relationen zwischen Knoten definieren, die als Funktion betrachtet werden können. Beispielsweise lässt sich für einen Elementknoten, der nicht die Wurzel des Dokuments ist, stets eindeutig der Elternknoten bestimmen, so dass diese *Eltern-von-Relation* als Funktion betrachtet werden kann. Bestimmte strukturelle Relationen sind in jedem XML-Dokument immer funktional, was im XML-Datenmodell verankert ist. So bestimmt ein Textknoten immer funktional den Elementknoten, der ihn enthält. Zudem hängen die Attribute eines Elementknotens immer funktional von diesem Element ab. In spezifischen XML-Dokumenten können unter Umständen weitere Relationen gefunden werden, die im jeweiligen Dokument eine Funktion darstellen. Ein Beispiel hierfür wäre ein Dokument, in dem jeder Pfad, der von einem Elementknoten von Typ b zum Wurzelknoten führt, über einen einzigen Elementknoten von Typ a geht. In einem solchen Dokument bestimmt ein jedes b-Element ein eindeutiges a-Element.

Bei der Instanziierung von EFGT-Netz-Eintragsschemata mit Hilfe von XML-Daten sollen funktionale Strukturrelationen aus der Baumrepräsentation des XML-Dokuments ausgewählt und auf funktionale Abhängigkeiten im Template abgebildet werden. Hierfür wird zur Extraktion von Tupeln aus einem XML-Dokument auf die XML-Abfragesprache XPath (vgl. XPath, 2006) zurückgegriffen. Variablen werden somit als XPath-Ausdrücke angegeben, die implizit Bezug aufeinander nehmen. Die Vorgehensweise hierbei sei anhand

des folgenden Beispiels erläutert.

A

```
<geo_data related="Switzerland">
  <canton name="Thurgau">
    <district name="Bezirk Weinfelden">
      <capital>Weinfelden</capital>
    </district>
    <district name="Bezirk Bischofszell">
      <capital>Bischofszell</capital>
    </district>
  </canton>
  <canton name="Wallis">
  ...
  </canton>
</geo_data>
```

B

```
<geo_data related="Switzerland">
  <canton name="Thurgau">
    <district id="3">
      <name>Bezirk Weinfelden</name>
      <capital>Weinfelden</capital>
    </district>
    <district id="4">
      <name>Bezirk Bischofszell</name>
      <capital>Bischofszell</capital>
    </district>
  </canton>
  <canton name= "Wallis">
  ...
  </canton>
</geo_data>
```

Abbildung 3.11: Zwei XML-Darstellungen der Daten in Tabelle 3.1

Abbildung 3.11 zeigt zwei Ausschnitte verschiedener XML-Dokumente, die beide die Daten in Tabelle 3.1 auf unterschiedliche Weise kodieren. Ausschnitt A kodiert die funktionalen Abhängigkeiten in der Tabelle mit strukturellen Relationen, die in einem XML-Dokument inhärent funktional sind: Der Textknoten "Weinfelden" bestimmt das Element vom Typ `capital` eindeutig, das seinerseits zu einem eindeutigen Elternknoten mit der Beschriftung `district` mit einem einzigen `name`-Attribut mit dem Wert "Bezirk Weinfelden" führt. Im Ausschnitt A bestimmt der Textknoten "Weinfelden" über eine komplexe Relation den Wert "Bezirk Weinfelden" eindeutig. Wie kann eine solche Relation in einem Template mit Hilfe von XPath spezifiziert werden?

Abbildung 3.12 zeigt eine auf Ausschnitt A abgestimmte Variante des Bezirkstemplates. Bei der Definition von Variablen fängt man damit an, mit Hilfe eines XPath-Ausdrucks, die Knotenmenge im XML-Dokument zu bestimmen, die die Werte für den äußersten Anker im Template darstellen. An dieser Stelle sei in Erinnerung gerufen, dass der äußerste Anker im Template alle anderen Werte im Template bedingt, da notwendigerweise alle weiteren Variablen im Skopus dieses Ankers liegen. Im Bezirkstemplate ist das der Anker `Hauptstadt`. Die Werte für diesen Anker können für Ausschnitt A mit dem XPath-Ausdruck `//capital` – der alle `capital`-Elemente im Dokument selektiert – ausgewählt und im Template dem Anker der Reihe nach zugewiesen werden. Da die anderen Variablen in Abhängigkeit davon belegt werden, welches `capital`-Element gerade betrachtet wird, werden alle anderen XPath-Verweise im Template in Bezug auf den Pfad ausgewertet, der beim aktuellen `capital`-Element startet und zur Dokumentwurzel führt. In XPath-Termini übersetzt bedeutet dies, dass die anderen Variablen auf der `ancestor-or-self`-Achse des aktuellen Knotens ausgewertet werden. Das wird implizit angenommen und braucht im Template nicht angegeben zu werden. Wenn vom `capital`-Element "Weinfelden" ausgegangen wird, extrahiert demzufolge der Ausdruck `district/@name` für den Anker `Bezirk` den Wert "Bezirk Weinfelden" sowie `canton/@name` für den Kanton den Wert "Thurgau".

Im Allgemeinen lässt sich die Vorgehensweise bei der Definition von Variablen für XML-Daten so zusammenfassen, dass zunächst für den äußersten Anker im Template ein XPath-Ausdruck angegeben wird, der die Menge der Knoten bestimmt, mit der der Anker instanziiert wird und als Schlüssel für die Belegung der anderen Variablen dient. Als nächstes werden die restlichen Variablen in der Reihenfolge "von außen nach innen" bezüglich des Aufbaus vom erweiterten ID-String aus definiert und zwar relativ zur `ancestor-or-self`-Achse der einzelnen Knoten in der Schlüsselmenge.

Diese Strategie ist sehr flexibel und erlaubt die Verarbeitung der verschiedenen Varianten, mit denen ein und dieselben funktionalen Relationen in einem XML-Dokument kodiert werden können. Dies ist auch der Fall, wenn die Kodierung der funktionalen Abhängigkeiten mit Hilfe struktureller Relationen erfolgt, die nur punktuell funktional für ein bestimmtes Dokument oder eine Dokumentenart sind, sogenannten *Pseudoattributen*. So kodiert im XML-Ausschnitt B (Abb. 3.12) ein Kind-Element des `capital`-Elements von Typ `name` den Namen des Bezirks, statt wie im Ausschnitt A ein Attribut des `capital`-Elements. Die Kind-von-Relation zwischen zwei Elementen ist in XML nicht inhärent funktional – das XML-Datenmodell erlaubt einem Element mehrere Kind-Elemente desselben Typs zu besitzen –, im XML-Dokument B aber durchaus, da jedes `capital`-Element genau ein `name`-Element umschließt. Das `name`-Element ist in diesem Dokument ein Pseudoattribut vom `capital`-Element. Relationen, die Pseudoattribute darstellen, können in einer DTD oder einem XML-Schema für das Dokument vorgegeben sein. Im Bezirkstemplate reicht aus, den Ausdruck `reference/@name` durch den Verweis `district/name` zu ersetzen, um den als Pseudoattribut angegebenen Namen des Bezirks zu erfassen und das Template korrekt zu instanziieren.

```
( Hauptstadt ( Bezirk g( Bezirksmenge G[#canton/@name].n).n)&[Hauptstädte])

  Bezirksmenge.name.de.nomen      = "Bezirke in #canton/@name"
  Bezirksmenge.variante.de.nomen  = "Bezirke #{canton/@name}s"
  Bezirk.name.de.nomen            = "#district/@name"
  Hauptstadt.name.de.nomen        = "#//capital"
```

Abbildung 3.12: Das *Bezirkstemplate* mit XPath-Variablen für den XML-Ausschnitt A in Abb. 3.11

3.4 Die Implementierung: Das *Upload-Tool*

Bisher wurde in diesem Kapitel eine spezielle Sprache vorgestellt, die als Integrationssprache bezeichnet wurde und eine abstrakte Lösung für die interne Anwendungslogik eines Systems darstellt, das zur Generierung und Alinierung von EFGT-Netz-Einträgen aus vorhandenen Daten dient. In diesem Abschnitt wird die technische Umsetzung dieser Sprache im Rahmen eines reellen Systems beschrieben. Dieses System, das *Upload-Tool*, geht

darüber hinaus auf das spezielle Ziel ein, die effiziente menschliche Kontrolle der automatisch generierten Einträge zu ermöglichen und den damit verbundenen, in Abschnitt 3.2 bereits betrachteten Anforderungen zu begegnen.

3.4.1 Allgemeiner Ablauf bei der Verwendung des Upload-Tools

Das Upload-Tool hat eine webbasierte Client-Server-Architektur. Im grundlegenden Ablauf des Hauptanwendungsfalls, der Integration von vorhandenen Daten, ruft der Benutzer in einem Webbrowser ein Formular auf, mit dem er ein EFGT-Netz-Eintragsschema und die dazugehörigen Daten hochladen kann. Als Antwort erscheint im Browser eine Darstellung des Templates und der hochgeladenen Daten, wodurch eine *Sitzung* beginnt. Im Verlauf dieser Sitzung können Eintragsschema und Daten überarbeitet und immer wieder ausgewertet werden, bis die generierten Einträge zufriedenstellend sind. Anschließend kann der Benutzer einzelne generierte Einträge auswählen und deren Übernahme in das EFGT-Netz bestätigen, wodurch erst die Wissensressource modifiziert wird. Zwischenzeitlich lassen sich Teile der bearbeiteten Daten exportieren, um sie in einer separaten Sitzung wieder zu verwenden oder zur Protokollierung.

Der allgemeine Ablauf einer Sitzung und die Funktionen, die dem Ontologieentwickler dabei zur Verfügung stehen, werden im Abschnitt *Steuerung des Integrationsprozesses im Client* (s. S. 95) näher betrachtet. Spezielle Abläufe, mit denen weitere Anwendungsfälle abgedeckt werden, werden im Abschnitt 3.4.5 beschrieben.

3.4.2 Architektur

Die Architektur des Upload-Tools folgt dem aus dem *Software-Engineering* bekannten Architekturmuster des *Model-View-Controllers (MVC)*. Ziel dieses Musters ist es, das Programm in drei Einheiten zu teilen: das Datenmodell (engl. *model*), das die Daten und meistens die Anwendungslogik enthält, die Präsentation (engl. *view*), d.h die Darstellung eines bestimmten Modells, auf der die Interaktion mit dem Benutzer stattfindet, und die Programmsteuerung (engl. *controller*), die die Benutzeraktionen entgegennimmt und entsprechende Änderungen des zugehörigen Datenmodells umsetzt.

Im Fall des Upload-Tools ist dieses Muster im Rahmen einer webbasierten Client-Server-Architektur umgesetzt, wodurch ein Szenario ermöglicht wird, in dem mehrere Ontologieentwickler verteilt an einem EFGT-Netz arbeiten. Abbildung 3.13 zeigt eine schematische Darstellung dieser Architektur.

Am Anfang der Benutzerinteraktion mit dem Upload-Tool wird das Programm serverseitig von einem Java-Servlet gesteuert, das die Sitzungen der verschiedenen Clients verwaltet. Es nimmt die vom Benutzer mittels des Formulars übertragenen Daten entgegen und baut eine neue Sitzung auf. Aus dem übertragenen Eintraggschema und den Daten wird zunächst das zugehörige Datenmodell aufgebaut, das ebenfalls in Java implementiert ist und im Server residiert. Das Datenmodell besteht aus folgenden Teilen:

- Das kompilierte *Template*, ein komplexes Objekt, das aus dem angegebenen Eintrags-

schema vom sogenannten *Interpreter* aufgebaut wird. Das kompilierte Template stellt eine Art Zusammenstellung abstrakter Einträge dar: Durch deren Auswertung und Belegung mit Daten entstehen dann generierte, konkrete Einträge, die als einzelne Objekte der Klasse *Concept* implementiert sind.

- Die *Originaldaten* werden intern als Objekte repräsentiert, die für jedes Format von entsprechenden *Parsern* aufgebaut werden. Dabei werden Tabellen und als typisierte Felder kodierte Daten als Objekte einer selbst entwickelten Klasse (*DataLines*) repräsentiert. XML-Daten werden intern als DOM-Datenstruktur gehalten, was mit Hilfe der Java-Bibliothek JDOM erfolgt. Den Klassen, die die Originaldaten halten, werden die im Template definierten Variablen übergeben, worauf die entsprechenden Variablenbelegungen aus der internen Datenstruktur extrahiert werden. Das Template wird dann auf Anforderung mit diesen Belegungen instanziiert.

- Der *Eintragsspeicher* stellt den erweiterten Eintragsraum aus Abschnitt 3.3.2 bereit, der die für die Alinierung nötigen Daten enthält. Im Eintragsspeicher werden alle generierten Einträge gehalten sowie alle im EFGT-Netz bestehenden Einträge, die für den Abgleich relevant sind. Eine performante Alinierung wird dadurch unterstützt, dass spezielle Indexstrukturen aufgebaut werden (s.u.). Der Eintragsspeicher ist zunächst leer und füllt sich im Laufe der Auswertung des Templates auf.

- Einzelne Objekte einer speziellen Klasse repräsentieren die *Alinierungsergebnisse*, d.h. die Ergebnisse der jeweiligen Instanziierungen des Templates. Ein solches Objekt fasst die Variablenbelegung, mit der das Template instanziiert worden ist, sowie resultierende Einträge oder Warnungen zusammen und dient als Grundlage für die Darstellung dieser Ergebnisse im Client.

Die Semantik der in Abschnitt 3.3 entwickelten Integrationssprache wird durch die im Datenmodell enthaltenen Objekte und ihr Verhalten abgedeckt. Dies bedeutet, dass die Anforderungen an die Anwendungslogik (s. S. 63) im Datenmodell realisiert sind.

Sind einmal die vom Benutzer übertragenen Daten geparst und das Datenmodell insgesamt aufgebaut, so wird die Präsentation der Daten als HTML-Seite generiert und an den Client zurückgeschickt. In dieser Darstellung, in der sowohl die Daten als auch das Eintragsschema repräsentiert und manipulierbar sind, kann der Benutzer diverse Aktionen ausführen. Während der Sitzung befindet sich das Steuerungsmodul für die Interaktion mit dem Datenmodell im Client und ist Javascript (JS) implementiert. Der Zugriff über JS auf die Java-Objekte des Datenmodells im Server erfolgt mit Hilfe einer speziellen AJAX Bibliothek für Java mit dem Namen *Direct Web Remotig* (vgl. DWR, 2008). DWR stellt einen *Remote Procedure Call*-Mechanismus (RPC) bereit, mit dem im Sinne des AJAX-Paradigmas asynchrone Requests des Clients an den Server abgesetzt werden können: DWR erzeugt automatisch für einzelne Methoden einer Java-Klasse korrespondierende JS-Funktionen, die im Client aufgerufen werden können. Ein spezielles Servlet nimmt von diesen JS-Funktionen gesendete Requests entgegen, ruft die passenden Methoden der

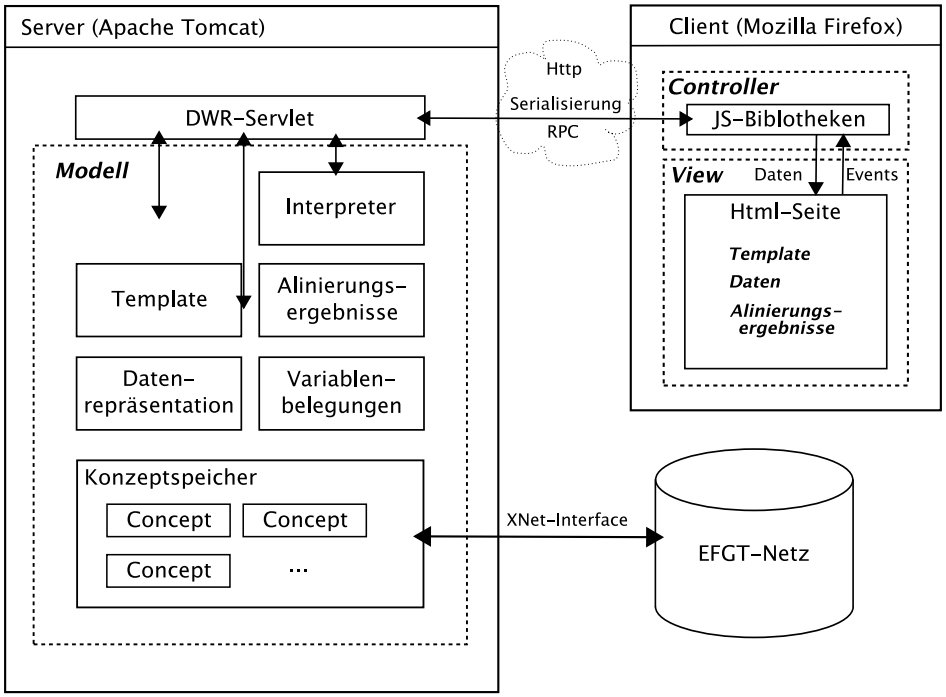


Abbildung 3.13: Schematische Darstellung der Architektur des Upload-Tools. Dargestellt sind die Komponenten, die während einer Sitzung aktiv sind.

entsprechenden Java-Objekte aus und gibt die Rückgabewerte zurück an den Client, der sie in die Darstellung einbindet.

Die Funktionalität und die Darstellung des Datenmodells im Client werden separat in Abschnitt 3.4.4 (s. S. 90 ff) beschrieben, da im Client die Interaktion mit dem Ontologieentwickler stattfindet und dort den Anforderungen bezüglich effizienter Steuerung und Überwachung nachgekommen werden muss.

Im Datenmodell sind besonders wichtige Komponenten der Interpreter und der Konzeptspeicher, da sie einen Großteil der Anwendungslogik zur Generierung und Alinierung von Einträgen bereitstellen.

Der Interpreter

Der Interpreter wurde mit Hilfe des Java Compiler Generators JavaCC implementiert und ist dafür zuständig, das kompilierte, instanziierbare Template aufzubauen. Die vom Interpreter verstandene Sprache für Eintragsschemata weicht von der in Abschnitt 3.3 verwendeten, abstrakten Syntax ab. Das gilt auch für die Darstellungen der Tabellen und typisierten Felder. Die konkrete Syntax, mit der Schemata und Daten zusammen in einem sogenannte *Upload-File* angegeben werden können, wird in Abschnitt 3.4.3 (s. S. 88 ff) beschrieben. Die Parser für Tabellen und typisierte Felder basieren auf regulären Ausdrücken, wofür auf die

entsprechende native Java-Bibliothek zurückgegriffen wurde. Zum Parsen von XML-Daten wird die bereits erwähnte Bibliothek JDOM eingesetzt. Diese stellt ebenfalls eine Implementierung von XPath bereit, die im Auswertungsmechanismus für die XML-spezifischen Verweise (vgl. S. 81 ff) genutzt wird.

Rolle des Eintragsspeichers bei der Alinierung

Die Alinierung der generierten Einträge mit dem EFGT-Netz erfolgt auf der Grundlage des Inhalts des Konzeptspeichers. Hierfür werden alle in derselben Sitzung generierten Einträge dieser Komponente hinzugefügt. Ebenfalls werden alle beim Abgleich relevanten Einträge aus dem EFGT-Netz geholt und im Speicher gehalten. Der Grund dafür ist, dass spezielle Indexstrukturen aufgebaut werden, um einen schnellen Zugriff auf die logische und die Attributrepräsentation aller im Speicher enthaltenen Einträge zu ermöglichen. Da die zu generierenden Einträge auf bestehenden aufbauen, wird üblicherweise innerhalb einer Sitzung mehrmals auf diese verwiesen. Durch die Aufnahme in den Eintragsspeicher und die dazugehörigen Indexstrukturen wird der Zugriff auf bestehende Einträge optimiert.

Der Eintragsspeicher ist die einzige Komponente, die mit der Wissensressource kommuniziert. Dies erfolgt über das vorhandene EFGT-Netz-Interface in seiner XML-Variante (s. Abschnitt 2.1.3, S. 40). Das bedeutet, das die nötige Information zur Auswertung des erweiterten ID-Strings im Template ebenfalls vom Konzeptspeicher bereitgestellt wird. Zur Ermittlung von Werten für die generischen Indizes im Template wird intern ein zusätzlicher Index von Teil-ID-Strings aufgebaut, die lokale Einführungen darstellen (s. S. 70 in Abschnitt *Semantik von EFGT-Netz-Eintragsschemata*). Die Auswertung von Query-Komponenten erfolgt mit Hilfe der internen Indexstrukturen und einer Implementierung der Funktion *lingSim* (s. ebd.), die im Grunde linguistische Angaben bzgl. der Groß-Kleinschreibung, den Trennern, usw. normalisiert. Dieselbe Funktion wird beim Abgleich der als Attribute kodierten linguistischen Repräsentation der Konzepte verwendet.

Da im Moment für jede Sitzung ein eigener Konzeptspeicher aufgebaut wird, können für den Benutzer unsichtbare Konflikte entstehen, wenn in mehreren parallelen Sitzungen mit ähnlichen Daten gearbeitet wird. In der momentanen Implementierung wird diese Art von Konflikten erst beim Hochladen der bestätigten Einträge bemerkt, in diesem Fall wurde also unnötige Arbeit in das Überprüfen der Einträge geleistet. Eine einfache Lösung zu diesem Problem wäre, Einträge des Netzes für andere Sitzungen zu sperren, solange auf sie in einer laufenden Sitzung zugegriffen wird. Um die kollaborative Arbeit auf denselben Einträgen zu ermöglichen, muss ein erweiterter Eintragsraum für alle parallel laufenden Sitzungen bereitgestellt werden. Dies würde bedeuten, dass Änderungen, die im Konzeptspeicher durch eine bestimmte Sitzung hervorgerufen werden, an die anderen bestehenden Sitzungen weitergegeben werden müssten und deren Alinierungsergebnisse betreffen könnten. Dadurch würde ein Observer-Architekturmuster umgesetzt, bei dem im Datenmodell auftretende Änderungen automatisch an die Präsentation weitergegeben würden und zu deren Aktualisierung führen würden. Wie bei Webanwendungen üblich, ist dieses Muster im Upload-Tool nicht realisiert worden. Dies wäre jedoch im Prinzip möglich, da die hier verwendete Bibliothek DWR ebenfalls unterstützt, dass Java-Objekte im Server

JS-Funktionen im Client aufrufen können, womit auftretenden Änderungen im Datenmodell an den Client weitergegeben werden könnten. Nach den bisherigen Erfahrungen stellt jedoch die Verwendung eines separaten Eintragsspeichers für jede Sitzung keine signifikate Einschränkung dar.

3.4.3 Eine konkrete Syntax für Templates und *Upload-Files*

Die bisher in diesem Kapitel aufgeführten Beispiele für EFGT-Netz-Eintragsschemata wurden in einer abstrakten Syntax angegeben, die sich durch die Verwendung typographischer Mittel wie etwa unterschiedlicher Schriftgrößen und -arten, tiefgesetzter Wörter, usw. besonders gut lesbar ist.

In der Implementierung lassen sich Eintragsschemata und Daten in einer Datei, dem sogenannten *Upload-File*, zusammenfassen, wobei hierfür eine Syntax verwendet wird, die zwar sehr nah an der bisher eingesetzten liegt, jedoch auf typographische Effekte verzichtet und auf der Unicode-Zeichensatzkodierung aufbaut.

Um das Parsen der Datei durch den Interpreter zu vereinfachen, wird die Datei durch die Verwendung spezieller Markierungen in verschiedene Abschnitte unterteilt. Das Aussehen dieser Markierungen erinnert an die Syntax von LaTeX-Steuerungsbefehlen. Jeder Abschnitt vom Typ *<Schlüsselwort>* beginnt mit einer Markierung der Form `\begin{`*<Schlüsselwort>*`}` und endet mit `\end{`*<Schlüsselwort>*`}`. Der Grundaufbau eines Upload-Files sieht folgendermaßen aus:

```
\begin{upload}
  \begin{program}
    ...
  \end{program}

  \begin{data}
   ...
  \end{data}
\end{upload}
```

Zeilenumbrüche und Leerzeichen außerhalb von Klammerungen und Angaben spielen im Format keine Rolle. Kommentare können außerhalb des `upload`-Bereichs angegeben werden.

Das Eintragsschema wird im `program`-Bereich angegeben, die Daten im `data`-Teil. Im `program`-Abschnitt können mehrere, mit dem Schlüsselwort `entries` markierte Abschnitte angegeben werden, die jeweils ein Eintragsschema enthalten. Nach dem Hochladen der Datei wird immer das Schema im ersten `entries`-Abschnitt ausgewertet; die anderen `entries`-Abschnitte dienen lediglich dazu, verschiedene Varianten des Schemas abzuspeichern. Ein `entries`-Abschnitt enthält den erweiterten ID-String und die Attributdeklarationen, die mit dem Schlüsselwort `names` gekennzeichnet werden. Im Template werden

Anker mit einem Ausdruck der Form `_{<Ankernamen>}` neben die entsprechende aufgehende Klammer des erweiterten ID-Strings gesetzt. Folgender `program`-Abschnitt stellt ein einfaches Schema mit einem Anker `NeuerEintrag` dar:

```
\begin{program}
\begin{entries}

  (_{NeuerEintrag}F([#Staat]&[Politik]).n)

  \begin{names}
   NeuerEintrag.name.de.nomen = "Politik in #Staat"
  \end{names}

\end{entries}
\end{program}
```

Die konkrete Syntax für Variablen stimmt mit der im Kapitel bisher verwendeten überein (s. Abschnitt 3.3.3). Dementsprechend stellt im Beispiel der Ausdruck `#Staat` einen Verweis auf eine Spalte einer Tabelle dar.

Im `data`-Abschnitt werden die Daten zur Instanziierung des Schemas angegeben. Die Syntax für Tabellen ist ebenfalls an der Tabellensyntax in LaTeX angelehnt. In einer Zeile werden im Normalfall die Zellen mit dem Zeichen `&` voneinander getrennt und die einzelnen Zeilen durch die Zeichenfolge `\\` abgeschlossen. Andere Trenner können ebenfalls verwendet werden, wobei sie beim Hochladen des Upload-Files im Formular entsprechend angegeben werden müssen. Die Spaltennamen werden im Upload-File unmittelbar nach der Marke `\begin{data}` als Liste deklariert:

```
\begin{data}{#Name|#Ort|#Synonym}
American Stock Exchange & New York & \\
Armenische Börse & Eriwan & Börse in Armenien \\
Augsburger Börse & Augsburg & \\
Australian Securities Exchange & Sidney & australische Wertpapierbörse \\
Berne eXchange & Bern & \\
Bolsa de Valores de Caracas & Caracas & Börse in Venezuela \\
Bombay Stock Exchange & Mumbai & \\
Börse Berlin & Berlin & \\
\end{data}
```

In der Liste werden die einzelnen Spaltennamen mit dem Zeichen `|` getrennt, wobei jedem das Zeichen `#` vorangestellt wird.

Bei typisierten Feldern werden die einzelnen Felder durch `;` getrennt; Typ und Wert eines Feldes durch `:`. Die Feldtypen, auf die die Variablen Bezug nehmen können, werden ähnlich wie Tabellenspalten spezifiziert, wobei in diesem Fall die Reihenfolge der Auflistung irrelevant ist und auf das Zeichen `#` verzichtet wird (Bsp. s. Abb. 3.20, S. 102).

Bei XML-Daten werden diese direkt zwischen die `data`-Markierungen gesetzt. Eine explizite Deklaration von Element- und Attributnamen im `data`-Abschnitt ist nicht nötig. Für das jeweilige Datenformat wird der entsprechende Parser beim Hochladen der Datei im Formular ausgewählt.

In Abschnitt 3.4.5 sind mehrere vollständige Beispiele von Upload-Files zu finden.

3.4.4 Der Client

Der Client ermöglicht die Interaktion des Ontologieentwicklers mit dem Upload-Tool und stellt hierfür die Funktionen bereit, mit Hilfe derer sich der Integrationsprozess steuern und überwachen lässt. Aus diesem Grund war ein explizites Ziel bei der technischen Umsetzung des Clients, die Kontrolle der automatisch generierten Einträge sowie die Analyse und Behebung auftretender Konflikte möglichst benutzerfreundlich zu gestalten, sodass der menschliche Aufwand selbst bei der Verarbeitung großer Datenmengen minimal gehalten wird.

Eine Leitidee dabei ist, dass die Kontrolle der Templateergebnisse sich besonders durch eine geeignete Visualisierung unterstützen lässt. Demnach sollte der Client ermöglichen, auf einen Blick die Effektivität des Eintragsschemas bezüglich der Generierung neuer Einträge einzuschätzen sowie Konflikte und problematische Datenbereiche schnell zu erkennen. Durch eine geeignete Darstellung der Alinierung und die Bereitstellung zusätzlicher Informationen sollten die Arbeitsweise des Templates nachvollziehbar und dessen Ergebnisse leicht interpretierbar werden.

Bei der Bereitstellung der Funktionen für den Benutzer wurde, neben der Unterstützung der grundlegenden Interaktion mit dem Template, das Hauptaugenmerk darauf gelegt, Standardlösungen für häufige Probleme zu entwickeln und innerhalb der Sitzung verfügbar zu machen. Insgesamt sollte dadurch die Effizienz einer Sitzung gesteigert und eine bessere Steuerung des Integrationsprozesses erreicht werden.

Visualisierung generierter Einträge und Alinierungsergebnisse

Die Visualisierung der Ergebnisse soll den Ontologieentwickler dabei unterstützen, die von einem bestimmten Template produzierten Ergebnisse nachzuvollziehen. Schließlich muss der Ontologieentwickler in der Lage sein, bei Bedarf das Template zu überarbeiten, um bessere Ergebnisse zu erzielen. Hierbei ist die Darstellung der Variablenbelegungen, mit denen das Template instanziiert wurde, ein wichtiger Anhaltspunkt. Bei Tabellen und typisierten Feldern, bei denen aus jeder Zeile eine einzige Variablenbelegung extrahiert wird, kann dies sehr leicht dadurch erreicht werden, indem die entsprechenden Spaltenzellen bzw. Felder farblich hervorgehoben werden. Die Hervorhebung von einzelnen Variablenbelegungen ist bei XML-Daten dagegen schwieriger, da in einem XML-Dokument eine bestimmte Angabe Bestandteil mehrerer Belegungen sein kann und dadurch letztere graphisch aufeinanderfallen können. In diesem Fall wird daher die Darstellung der einzelnen Belegungen von der Darstellung der Originaldaten getrennt: Eine Auflistung aller verwendeten Belegungen erfolgt in einem separaten Bereich; eine Verbindung zu den Originaldaten wird

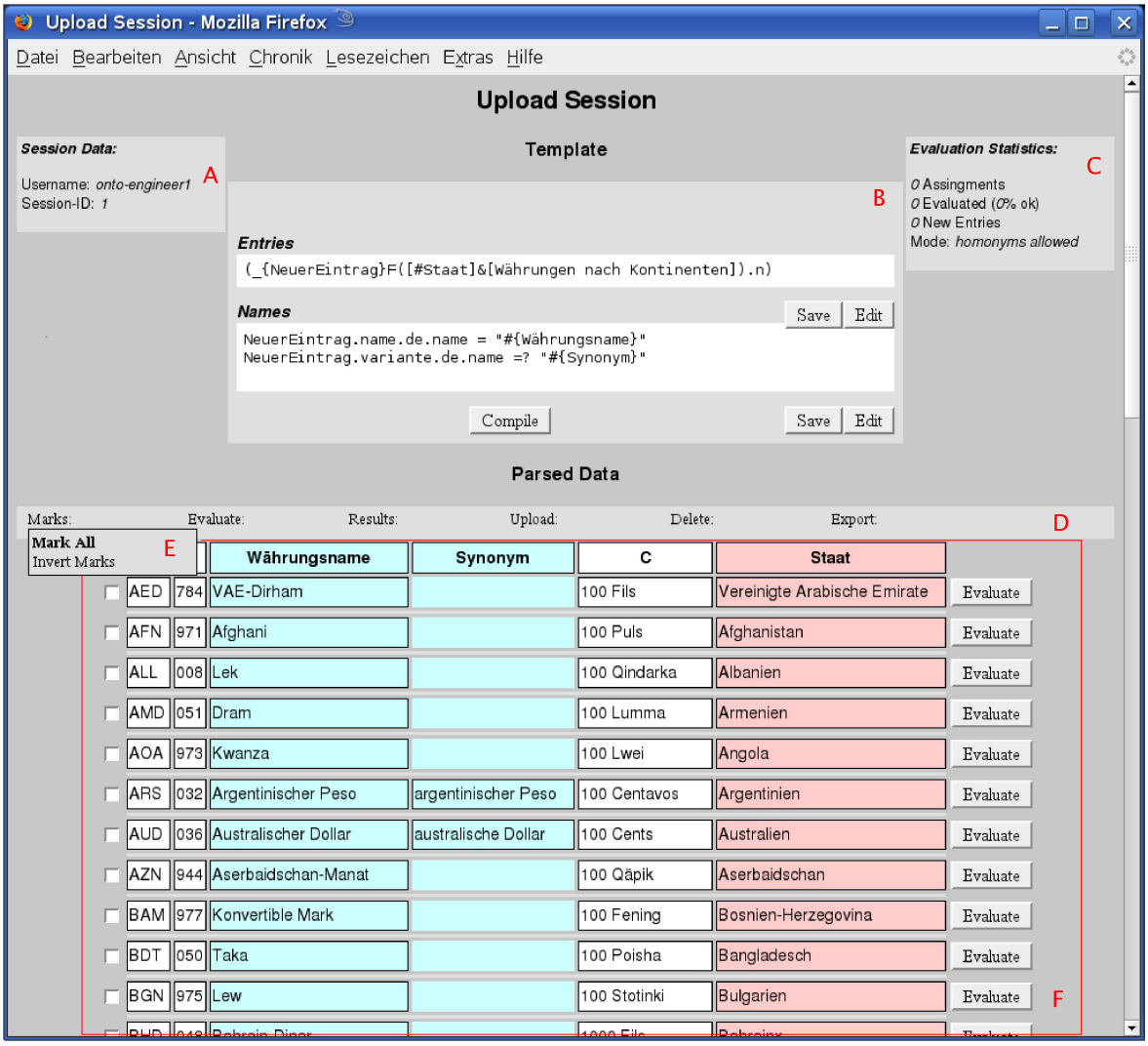


Abbildung 3.14: Der Client des Upload-Tools. Bereiche: A: Sitzungsdaten. B: Editieren und Kompilieren des Templates. C: Statistiken über Alinierungsergebnisse. D: Reiter mit verschiedenen Funktionenmenüs. E: Aufgeklapptes Funktionenmenü zur Markierung aller Instanziierungen bzw. zur Invertierung der Markierung. F: Darstellung der Daten, hier eine Tabelle. Auf hellblau unterlegte Spalten wird im *names*-Teil des Template verwiesen, auf hellrote im ID-String-Muster.

jedoch dadurch behalten, dass das Anklicken einer bestimmten Belegung die Hervorhebung der entsprechenden Angaben im XML-Dokument auslöst. Diese Beziehung zu den Originaldaten ist wichtig, da sie dem Ontologieentwickler als Orientierung bei der Definition von XPath-Variablen dient.

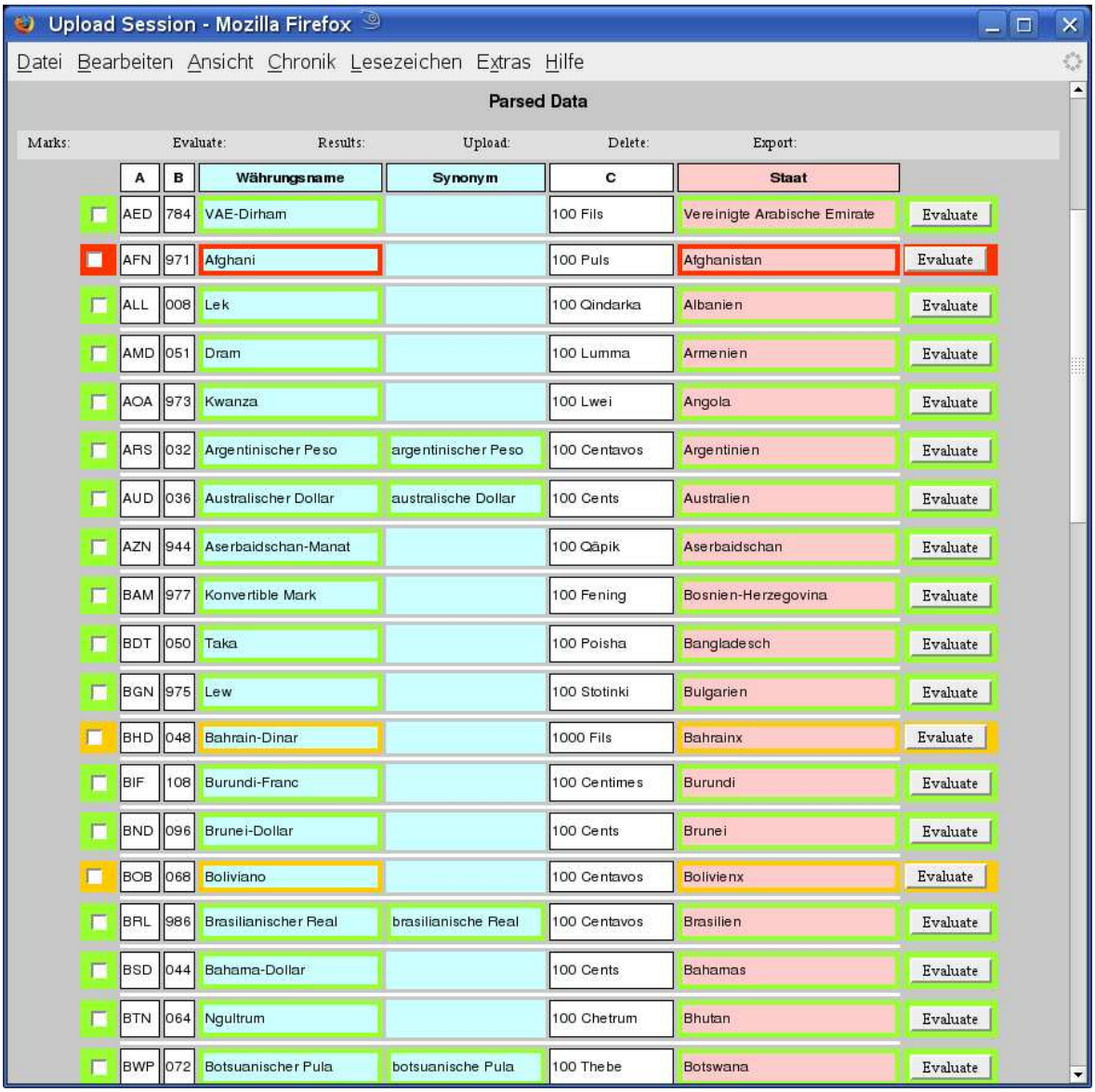
Ein erster Eindruck, wie gut die Integration der Daten mit dem angegebenen Template funktioniert, lässt sich vermitteln, indem die einzelnen Variablenbelegungen mit unterschiedlichen Farben umrandet werden – je nachdem, ob das Ergebnis der entsprechenden Instanziierung ein neuer Eintrag ist, einen Konflikt darstellt oder auf Grund von Auswertungsfehlern insgesamt nicht ermittelt werden konnte. Abbildung 3.15 zeigt ein Beispiel dieser globalen Sicht, mit der die Tauglichkeit des Templates optisch eingeschätzt werden kann. Zusätzlich informiert eine Statistik über die Ergebnisse der Templateauswertung.

Für die einzelnen Variablenbelegungen kann das genaue Ergebnis der Templateauswertung in der Oberfläche aufgeklappt werden. Konnte das Template mit der Belegung erfolgreich ausgewertet werden, so werden die generierten Einträge als Baum dargestellt, der dem Aufbau des erweiterten ID-Strings im Template entspricht (s. Abb. 3.16). Die Knoten im Baum stellen Konzepte des EFGT-Netzes dar, die mit ihrer Vorzugsbezeichnung repräsentiert werden; die Kanten im Baum korrespondieren mit den lokalen Einführungen und die Verwendung des `&`-Operators im Template. Durch einen *mouse-over*-Effekt wird die komplette Attributrepräsentation eines Konzeptes angezeigt, wobei die eingesetzten Daten aus der Variablenbelegung besonders markiert sind. Die Darstellung der Konzepte ist anklickbar, um in einem separaten Fenster die Navigationsoberfläche erscheinen zu lassen und die Position des Konzepts im Netz überprüfen zu können.

Die Alinierungsergebnisse sind in der Repräsentation der generierten Einträge graphisch integriert. Einträge, deren Vorzugsbezeichnung in roter Schrift angegeben ist, sind im EFGT-Netz vorhandene Einträge. Blaue Schrift steht für Einträge, die in derselben Sitzung aufgrund einer anderen Instanziierung erzeugt wurden; grüne Schrift stellt neue Einträge der aktuellen Variablenbelegung dar. Im Fall neuer Einträge werden eventuell vorhandene zusätzliche Informationen aus den approximativen Matchingverfahren (Fälle mit Code 20, 02, 22 in Tab. 3.2, S. 72) unterhalb der Eintragsdarstellung als Liste von anklickbaren Konzepten angezeigt. Das heißt praktisch, dass ggf. entsprechende lokale Einführungen als partielle logische Matches aufgeführt werden, bei denen eine zusätzliche Überlappung der linguistischen Repräsentation gegeben ist (s. Abschnitt *Zusätzliche Information durch partielles Matching*, S.73 ff).

Im Fall eines logischen oder linguistischen Konflikts (Fall 01 bzw. 10) wird der Baum entsprechend der Auswertungsreihenfolge des erweiterten ID-Strings bis zum ersten auftretenden Konflikt aufgebaut. Eine eventuell vorhandene Zusatzinformation zum Konflikt (Fälle 21 bzw. 12 in Tab. 3.2) wird ebenfalls ausgegeben.

Kann für eine bestimmte Variablenbelegung das Template nicht ausgewertet werden, weil etwa der Wert für eine bestimmte Variable fehlt, eine Multivariable nicht mindestens einen Wert enthält usw., wird der Grund als Hinweis an den Benutzer ausgegeben. Im Fall einer Query-Komponente, die sich bei der Auswertung als ambiger Verweis auf mehrere Konzepte herausstellt, werden dem Benutzer die entsprechenden Konzepte aufgelistet.



A	B	Währungsname	Synonym	C	Staat	
AED	784	VAE-Dirham		100 Fils	Vereinigte Arabische Emirate	Evaluate
AFN	971	Afghani		100 Puls	Afghanistan	Evaluate
ALL	008	Lek		100 Qindarka	Albanien	Evaluate
AMD	051	Dram		100 Lumma	Armenien	Evaluate
AOA	973	Kwanza		100 Lwei	Angola	Evaluate
ARS	032	Argentinischer Peso	argentinischer Peso	100 Centavos	Argentinien	Evaluate
AUD	036	Australischer Dollar	australische Dollar	100 Cents	Australien	Evaluate
AZN	944	Aserbaidschan-Manat		100 Qäpik	Aserbaidschan	Evaluate
BAM	977	Konvertible Mark		100 Fening	Bosnien-Herzegovina	Evaluate
BDT	050	Taka		100 Poisha	Bangladesch	Evaluate
BGN	975	Lew		100 Stotinki	Bulgarien	Evaluate
BHD	048	Bahrain-Dinar		1000 Fils	Bahrainx	Evaluate
BIF	108	Burundi-Franc		100 Centimes	Burundi	Evaluate
BND	096	Brunei-Dollar		100 Cents	Brunei	Evaluate
BOB	068	Boliviano		100 Centavos	Bolivienx	Evaluate
BRL	986	Brasilianischer Real	brasilianische Real	100 Centavos	Brasilien	Evaluate
BSD	044	Bahama-Dollar		100 Cents	Bahamas	Evaluate
BTN	064	Ngultrum		100 Chetrum	Bhutan	Evaluate
BWP	072	Botsuanischer Pula	botsuanische Pula	100 Thebe	Botswana	Evaluate

Abbildung 3.15: Globale Sicht auf Alinierungsergebnisse. Farbliche Unterlegung der Zeilen entsprechend des Auswertungsergebnisses: grün: konfliktfrei; rot: konflikt; orange: fehlgeschlagen.

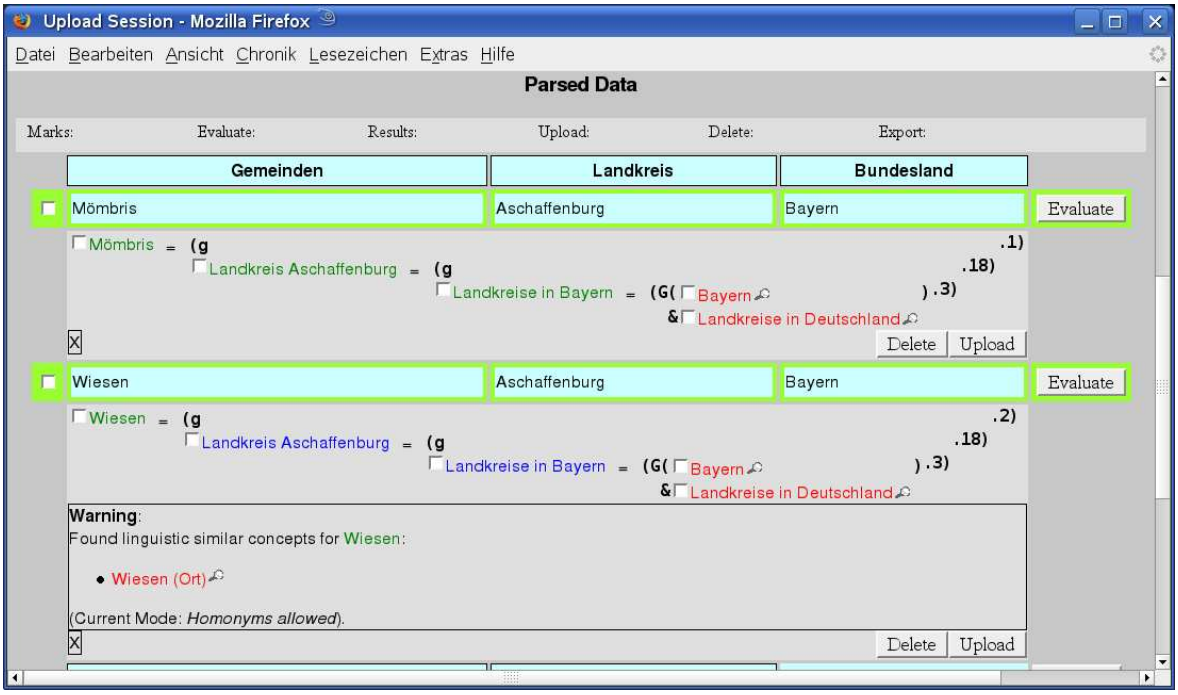


Abbildung 3.16: Darstellung einzelner, konfliktfreier Alinierungsergebnisse

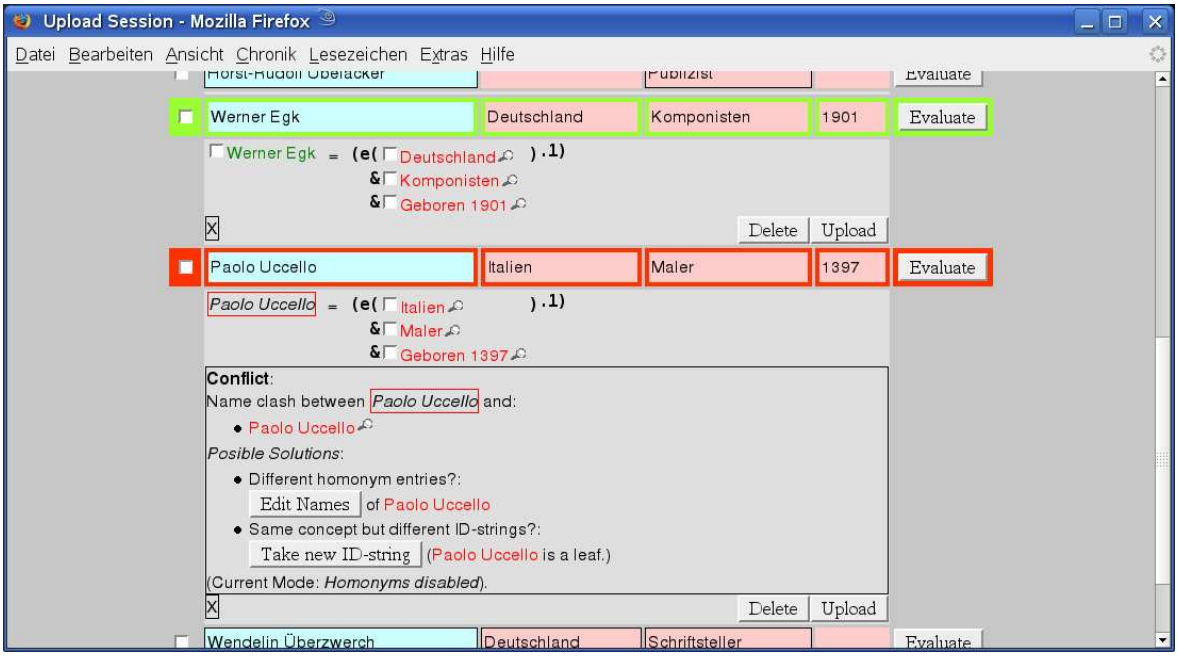


Abbildung 3.17: Darstellung von Konflikten

Steuerung des Integrationsprozesses im Client

Nachdem das entsprechende Upload-File hochgeladen wurde, läuft eine typische Sitzung folgendermaßen ab, wobei hierbei verschiedene, vom Client bereitgestellte Funktionen zum Einsatz kommen:

1. Das im Upload-File angegebene Template wird für alle erzeugbaren Variablenbelegungen ausgewertet.
2. Wird dadurch eine signifikante Anzahl von neuen Einträgen generiert – dies kann etwa in der vorher beschriebenen globalen Sicht überprüft werden – kann zu Schritt 3 übergegangen werden. Ansonsten kann der Benutzer versuchen, das Eintragsschema zu editieren und effektiver zu machen.
3. Erfolgreich erzeugte neue Einträge werden bestätigt und dem EFGT-Netz hinzugefügt. Optional kann eine spezielle, als Protokoll dienende Kopie des Upload-Files erstellt werden (s.u.).
4. Leicht lösbare Konflikte sowie behebbare, auf Grund der Daten verursachte Auswertungsfehler werden behandelt und das Template neu ausgewertet. Anschließend kehrt man zurück zu Schritt 3 oder geht über zum nächsten Schritt.
5. Erstellung einer sogenannten Differenzdatei: Bestimmte Daten können in ein separates Upload-File ausgelagert werden, um sie später händisch oder mit einem anderen Schema zu bearbeiten.

Dieser Ablauf stellt nur eine bestimmte Möglichkeit dar, die vom Client bereitgestellten Funktionen zu verwenden. Im folgenden werden diese näher betrachtet, deren Einsatz im verschiedenen Anwendungsfällen wird in Abschnitt 3.4.5 beschrieben.

Editierung, Kompilierung und Auswertung des Eintragsschemas: Prinzipiell wird der Integrationsprozess wird dadurch gesteuert, dass das Eintragsschema angegeben und für bestimmte Daten ausgewertet wird.

Im Client kann das im Upload-File angegebene Eintragsschema editiert und neu kompiliert werden. Dies erfolgt mit Hilfe der konkreten Syntax, wobei eventuelle Fehler in der Angabe an den Benutzer zurückgemeldet werden. Wurde das Schema erfolgreich kompiliert, kann es instanziiert und ausgewertet werden. Der Benutzer kann entscheiden, ob bei der Auswertung alle möglichen Belegungen für die im Template angegebenen Variablen eingesetzt oder ob nur bestimmte, ausgewählte Variablenbelegungen – insbesondere eine einzige – verwendet werden sollen.

Zur Behebung von Problemen bei der Auswertung können hochgeladene Daten in einzelnen Fällen ebenfalls editiert werden (s.u.).

Konfliktbehandlung: Die von der Oberfläche angebotenen Funktionen ermöglichen in vielen Fällen die Behandlung auftretender Alinierungskonflikte innerhalb der Sitzung. Dadurch soll die Effektivität der Sitzung gesteigert und die Tatsache ausgenutzt werden, dass die Sitzung einen Kontext darstellt, in dem sich Konflikte leichter interpretieren lassen. Schließlich wird durch die Alinierung einer bestimmten semantischen Klasse oder Hierarchie, die in den Daten repräsentiert ist, aus einem bestimmten Blickwinkel auf das EFGT-Netz zugegriffen.

Das Ziel ist, leicht lösbare und leicht zu verstehende Konflikte *on the fly* während des Integrationsprozesses beheben zu können. In der Regel kann ein Konflikt dadurch aufgelöst werden, dass der in Konflikt stehende, im Netz vorhandene Eintrag modifiziert wird. Um dies innerhalb der Sitzung zu ermöglichen, kann je nach Fall die Kodierungsoberfläche (s. Abschnitt 2.2.2) mit einem speziellen Knopf aufgerufen werden. Diese erscheint dann in einem separaten Fenster, wobei das Eintragsformular entsprechend der Lösung des Konflikts vorbelegt wird (s.u.). Der Benutzer braucht nur den modifizierten Eintrag zu bestätigen, kann aber auch zusätzliche Änderungen vornehmen. Im einzelnen werden vom Upload-Tool folgende Funktionen zur Behandlung von Konflikten angeboten (vgl. die in Tab. 3.2, S. 72 angegebenen Aktionen):

- Namenskonflikte (Fall 10)
  - Will man bei homonymen Konzepten (1. Interpretation) vermeiden, eine Ambiguität in der Wissensressource einzuführen, kann die Kodierungsoberfläche aufgerufen werden, um die linguistische Repräsentation des vorhandenen Eintrags zu ändern, etwa dessen Vorzugsbezeichnung. Alternativ kann bei den Formaten “Tabelle” und “typisierte Felder” der entsprechende Wert direkt in der Datenrepräsentation editiert werden.
  - Modelliert der generierte Eintrag ein vorhandenes Konzept logisch anders als der gefundene ID-String (2. Interpretation) kann die Behandlung des Konflikts meistens nicht innerhalb der Sitzung stattfinden. Dies ist nur dann möglich, wenn es sich beim vorhandenen Eintrag bezüglich der Struktur des EFGT-Netzes um ein Blatt handelt: Die Änderung des vorhandenen ID-Strings muss ansonsten an die Nachfahren des vorhandenen Eintrags weitergegeben werden. Dies stellt ein aufwändiges Verfahren dar, das mit Hilfe des Rename Tools (s. Abschnitt 2.2.2) außerhalb der Sitzung erledigt werden muss. Das gilt ebenfalls, wenn man die Modellierung des generierten Eintrags übernehmen will. Nur für den Fall eines Blattes wird ermöglicht, in der Kodierungsoberfläche den vorhandenen ID-String durch den generierten zu ersetzen.
  - In der momentanen Implementierung kann Fall 12 (Namenskonflikt mit partiellem logischen Match) nicht beobachtet werden, da der approximative logische Abgleich sich darauf beschränkt, lokale Einführungen, die sich nur im Index unterscheiden, als ähnlich zu erkennen. Wäre in diesem Sinne der zu generierende Eintrag einem schon vorhandenen mit derselben linguistischen Repräsentation

logisch ähnlich gewesen, wären sie bereits bei der Templateauswertung miteinander aliniert worden. Eine Art des approximativen logischen Abgleiches, die ermittelt, ob einer der verglichenen ID-Strings im Sinne zusätzlicher Komponenten eine Verfeinerung der Modellierung des anderen darstellt, wäre besonders interessant. Damit wäre für den Ontologieentwickler sofort erkennbar, in welchen Fällen der generierte ID-String eine Aktualisierung des vorhandenen darstellt und ob er übernommen werden sollte. Dies könnte mit einer geeigneten Visualisierung stark unterstützt werden.

Aus Effizienzgründen wurden zwei Ausführungsmodi realisiert, die Namenskonflikte ignoriert oder deren Behandlung erzwingt. In Abhängigkeit der vorliegenden Daten kann die eine oder andere Strategie relevant sein: Bei Toponymen etwa, bei denen homonyme Namen durchaus üblich sind, ist die Zulassung von Homonymie bei der Erzeugung von Einträgen in der Wissensressource sinnvoll. Bei Personennamen eines Bereichs kann oft davon ausgangen werden, dass es keine Homonyme gibt, sodass der Modus, der die Behandlung der Namenskonflikte erzwingt, zur systematischen Überprüfung der generierten Einträge genutzt werden kann.

- Logische Konflikte (Fall 01)
  - Ergänzt die Attributrepräsentation des generierten Eintrags die des vorhandenen (1. Intrepretation), kann das Formular des vorhandenen Eintrags aufgerufen und dann mit den neuen zusätzlichen Attributen ergänzt werden.
  - Fällt der ID-String des vorhandenen und des generierten Eintrags zusammen und sind dabei zwei semantisch verschiedene Konzepte gemeint (2. Interpretation), so ist die Behandlung des Konflikts *on the fly* nicht immer möglich. Dies hängt davon ab – ähnlich wie bei der 2. Interpretation eines Namenskonflikts –, ob der vorhandene Eintrag in der Struktur des Netzes ein Blatt darstellt oder nicht. Dementsprechend wird die Modifikation des vorhandenen ID-Strings in der Kodierungsoberfläche nur bei Blatteinträgen zugelassen.
  - Ist der generierte Eintrag dem vorhandenen zudem linguistisch ähnlich, (Fall 21), lässt sich der Konflikt wie bei der ersten Interpretation durch Überarbeitung der Attribute des vorhandenen Eintrags im Formular leicht lösen.

Im Fall eines generierten Eintrags, der potenziell neu ist, für den aber zusätzliche partielle Matches ermittelt wurden, können ähnliche Funktionen denen zur Konfliktbehandlung nützlich sein. In der aktuellen Implementierung bedeutet der Fall 22 aus Tab. 3.2, dass eine ähnliche lokale Einführung mit einem ähnlichen Namen zum generierten Eintrag in der Ressource gefunden wurde. In diesem Fall will man entweder den generierten Eintrag verwerfen oder die linguistische Repräsentation des vorhandenen Eintrags überarbeiten, was durch Aufruf der Kodierungsoberfläche möglich ist. Ähnlich kann Fall 20 behandelt werden. Fall 02 liefert in der momentanen Implementierung lediglich Informationen über ähnliche lokale Einführungen.

Nach der Behandlung des Konflikts kann die entsprechende Variablenbelegung neu ausgewertet werden, wodurch die Alinierung neu versucht wird. Taucht dabei ein neuer Konflikt auf, lässt er sich wie beschrieben behandeln.

Behebung anderer Probleme bei der Templateauswertung: Bei der Auswertung des Eintragsschemas kann es vorkommen, dass bei einer bestimmten Instanziierung nicht alle Variablen mit einem Wert aus den Daten belegt werden können, etwa weil in einer bestimmten Zeile einer Tabelle kein Wert für die entsprechende Spalte vorhanden ist. Die Sprache der Templates stellt verschiedene Mittel zur Verfügung (optionale Attribute, bedingte Anweisungen, variable Summen), mit denen sich solche Fälle behandeln lassen. Ist die Editierung des Templates nicht sinnvoll, können die betroffenen Belegungen einfach markiert und von der Auswertung ausgenommen werden.

Datensätze, die in Query-Komponenten zum Verweis auf vorhandene Konzepte eingesetzt werden und kein Ergebnis liefern, lassen sich im Client bei linearen Formaten (Tabellen, typisierten Feldern) direkt editieren. Angedacht ist, bei der Editierung einer Tabellenzelle oder eines typisierten Felds eine Suchfunktion in der linguistischen Repräsentation der im EFGT-Netz vorhandenen Einträge bereitzustellen, damit ein sinnvoller Ersatz für den fehlerhaften Datensatz gefunden werden kann. Bei Query-Komponenten kann es auch vorkommen, dass der verwendete Verweis ambig ist und mehreren Konzepten im Netz entspricht. Beim Vorkommen dieser ambigen Referenzen wird dem Benutzer angeboten, zwischen einer der verschiedenen Interpretationen auszuwählen, woraufhin die Auswertung des Schemas neu versucht werden kann.

Überwiegen bei der Auswertung des Templates diese Art der Probleme, stellt das Editieren oder die neue Definition des Schemas oft eine sinnvolle Maßnahme dar.

Protokollierung der Sitzung: Serverseitig werden Sitzungen protokolliert, indem laufend verschiedene Daten festgehalten werden. Dazu gehört der angemeldete Benutzer, das Datum, usw., vor allem aber die der Wissensressource hinzugefügten generierten Einträge sowie jeweils eine Kopie der Variablenbelegung und des dafür verwendeten Eintragsschemas. Damit lassen sich im Notfall Änderungen zurücknehmen.

Im Client steht dem Benutzer eine *Exportfunktion* zur Verfügung, mit der eine zusätzliche Art der Protokollierung ermöglicht wird. Die Exportfunktion erzeugt ein neues Upload-File mit dem aktuellen Zustand des Templates und der Daten. Besonders interessant ist, ein solches "Bild der Daten" zu machen, unmittelbar nachdem mit einem bestimmten Schema Einträge generiert und bestätigt wurden. Dieses neue Upload-File kann seinerseits hochgeladen werden und als Grundlage für eine neue Sitzung benutzt werden, in der die bereits durchgeführten Einträge neu aufgebaut und mit sich selbst abgeglichen werden. Im Rahmen dieser Realinierung werden die Einträge wiedergefunden oder es entstehen Konflikte, weil zwischenzeitlich einzelne Einträge verändert wurden. Dies stellt eine einfache Möglichkeit dar, Änderungen bei semiautomatisch integrierten Daten zu verfolgen.

Es ist geplant, die Exportfunktion so zu erweitern, dass die Identifikationsnummer der entsprechenden Einträge in der Wissensressource in das exportierte Upload-File integriert

wird. Durch die direkte Alinierung der Daten im Upload-File mit Einträgen in der Ressource über diese Identifikationsnummer erfolgt eine Art modularer Zugriff auf Einträge, der dafür genutzt werden kann, nachträgliche Änderungen zu beobachten, Einträge zu löschen, in ihrer Attributrepräsentation zu erweitern, usw..

Erzeugung von Differenzdateien: Eine andere Verwendung der Exportfunktion besteht darin, bestimmte Zeilen (Formate "Tabelle" und "typisierte Felder") bzw. Variablenbelegungen (Format "XML") auszuwählen und in ein separates Upload-File auszulagern. Dabei besteht die Option, das aktuelle Schema mit zu übernehmen[1]. Eine solche Differenzdatei zu erzeugen kann in verschiedenen Fällen von Nutzen sein; so können für Konflikte, die sich nicht innerhalb der Sitzung leicht lösen lassen, die entsprechenden Datensätze exportiert werden, um sie separat händisch zu bearbeiten. Darüber hinaus können Variablenbelegungen in einer Differenzdatei zu einer Klasse zusammengefasst werden, wenn bei der Betrachtung von Alinierungsergebnissen auffällt, dass bestimmte logische Konflikte systematisch vorkommen und sich die entsprechenden Datensätze besser mit einem anderen Template behandeln lassen.

3.4.5 Anwendungsfälle

Population der Ontologie mit facettierten Klassen

Bei der Beschreibung der *Steuerung des Integrationsprozesses im Client* (s. S. 95) wurde der Ablauf einer Sitzung skizziert, der die Vorgehensweise bei der Population der Ontologie mit voneinander unabhängigen, facettierten Einträgen gut beschreibt. Bei der Integration von Daten, die eine bestimmte Klasse von Elementen darstellen und nicht aufeinander aufbauen, kann der Integrationsprozess besonders flexibel gestaltet werden. Eine sinnvolle Strategie besteht darin, alle generierten Einträge eines produktiven Eintragsschemas zu übernehmen und erst dann auftretende Probleme zu behandeln. In vielen Fällen lassen sich Teile der problematischen Variablenbelegungen zu Klassen zusammenfassen, die sich mit einem eigenen Schema behandeln lassen. Danach kann anhand der übrigbleibenden Konflikte entschieden werden, inwiefern die Bearbeitung der Sonderfälle für eine zufriedenstellende Abdeckung der Klasse notwendig ist.

Wird festgestellt, dass eine Facette von der Wissensressource schlecht abgedeckt wird, etwa verschiedene Berufe für die Kodierung von Personen, so ist es sinnvoll, eine Differenzdatei zu erstellen, die die Zeilen bzw. Variablenbelegungen enthält, bei denen sich Lücken in der Facette auftun. Diese kann dann im Rahmen der Wissensakquisition gezielt ergänzt werden.

Abb. 3.18 zeigt den Ausschnitt eines Upload-Files, das zur Population der Ontologie mit Personen verwendet wurde. Im Eintragsschema wird eine bedingte Anweisung verwen-

[1] Im Fall von XML werden die ausgewählten Variablenbelegungen als Tabelle mit durchnummerierten Spalten exportiert. Wird das Eintragsschema ebenfalls in die Datei übernommen, so werden die als Variable angegebenen XPath-Ausdrücke durch Verweise auf die entsprechenden Tabellenspalten ersetzt, um die Verwendbarkeit der Differenzdatei als Upload-File zu gewährleisten.

det, mit der eine unterschiedliche Kodierung einer Person erzeugt wird, je nachdem, ob für die Spalte `year` ein Wert vorhanden ist. Es zeigt ebenfalls eine noch nicht vorgestellte syntaktische Variante der variablen Summe, die wie üblich durch `&*` repräsentiert wird, in der jedoch die eingebettete Query-Komponente keine Multivariable enthält. Die Auflistung von Spaltenvariablen, die durch den Balken `|` getrennt werden, stellt eine einfache Möglichkeit dar, bei Tabellen auf eine unterschiedliche Anzahl von belegten Spalten zuzugreifen. Ist in einer Zeile kein Wert sowohl für `nation` als auch für `rolle` definiert, so schlägt das Schema für diese Zeile fehl.

```
\begin{upload}  \begin{program} \begin{entries}

    if #year != '' then
        (_{Person}e(&*[#nation|#rolle]&[Geboren #year]).n)
    else
        (_{Person}e&*[#nation|#rolle].n)
    endif
    \begin{names}
      Person.name = "#name"
    \end{names}

\end{entries}
\end{program}

\begin{data}{#name|#nation|#rolle|#year}
Otto Ubbelohde         & Deutschland & Maler &   \\
Horst-Rudolf Übelacker &             & Publizist &   \\
Paolo Uccello          & Italien     & Maler & 1397 \\
Hermann Uber           & Deutschland & Komponisten &  \\
Wendelin Überzwerch    & Deutschland & Schriftsteller &  \\
Jorge Ubico Castañeda  & Guatemala   & Präsident &  \\
Heinrich Übleis        & Österreich  & Juristen &Evaluate  1993 \\
...
```

Abbildung 3.18: Upload-File zur Population der Ontologie mit Personen

Eine im Deutschen interessante Klasse von facettierten Einträgen bilden Komposita. Abb. 3.19 stellt ein Upload-File dar, mit dem auf dem Stamm *-tee* aufbauende Komposita kodiert und in die Ontologie integriert werden können. Im Beispiel wird auch die Bedeutung des Eintragsschemas zur effizienten Erzeugung graphischer und Flektionsvarianten sichtbar. Die systematische Aufbereitung von Komposita-Upload-Files setzt den Einsatz von Programmen zur Extraktion und Segmentierung von Komposita in ihre Bestandteile in der Wissensakquisitionsphase voraus.

```
\begin{upload} \begin{program} \begin{entries}

    (_{NeuerEintrag}F([#AusWas]&[Teearten]).n)

    \begin{names}
      NeuerEintrag.name.de.nomen = "#{Name}tee"
      NeuerEintrag.variante.de.nomen = "#{Name}tees"
      NeuerEintrag.variante.de.nomen = "#{Name}-Tee"
      NeuerEintrag.variante.de.nomen = "#{Name}-Tees"
      NeuerEintrag.variante.de.nomen =? "#{Synonym}tee"
      NeuerEintrag.variante.de.nomen =? "#{Synonym}tees"
      NeuerEintrag.variante.de.nomen =? "#{Synonym}-Tee"
      NeuerEintrag.variante.de.nomen =? "#{Synonym}-Tees"
    \end{names}
\end{entries}
\end{program}
\begin{data}{#AusWas|#Name|#Synonym}
Kamille  & Kamillen &  \\
Pfefferminze & Pefferminz & \\
Fenchel & Fenchel & \\
Salbei & Salbei & \\
...
```

Abbildung 3.19: Upload-File zur Population der Ontologie mit Komposita

Ausbau des EFGT-Netzes mit Hierarchien

Bei der Integration einer Hierarchie, die im Format typisierte Felder kodiert wurde, ist der Ablauf der Sitzung weniger flexibel als bei der Population mit voneinander unabhängigen Einträgen. Ein Unterschied zu diesem Anwendungsfall ergibt sich schon vor Beginn der Sitzung bei der Aufbereitung des Upload-Files: Es muss überprüft werden, ob das Upload-File tatsächlich eine Hierarchie kodiert und die Anforderungen an die Wohlgeformtheit der Datei erfüllt sind, die in Abschnitt *Kodierung von Hierarchien mittels typisierter Felder*, S. 79 ff beschrieben wurden. Diese Prüfung erfolgt mit Hilfe des im selben Abschnitt angesprochenen, externen Programms, das u.a. die Zeilen im Datenteil des Upload-Files korrekt sortiert. Ebenfalls müssen eventuell fehlende vorausgesetzte Einträge, an die die Hierarchie im EFGT-Netz anknüpft, vorab eingearbeitet werden.

Da die Einträge der Hierarchie aufeinander aufbauen, müssen dann innerhalb der Sitzung auftretende Konflikte in der Reihenfolge behandelt werden, die der Sortierung der Zeilen entspricht. In der Regel deckt das Schema alle Einträge der Hierarchie und braucht nicht während der Sitzung editiert zu werden. Ein Beispiel ist in Abb. 3.20 dargestellt.

```
\begin{upload} \begin{program} \begin{entries}

        (_{Krankheit}F&*[#*kat].n)

        \begin{names}
        Krankheit.name.de.nomen = "#mainDE"
        \end{names}
\end{entries}
\end{program}

\begin{data}{mainDE|kat|de|en|lt}
mainDE:Infektionskrankheiten;kat:Mikrobioloige;kat:Krankheiten nach Typ; \\
mainDE:Parasitäre Infektion;de:Parasiteninfektion;
            en:parasitic infections;kat:Parasitologie;kat:Infektionskrankheiten;\\
mainDE:Wurminfektionen;kat:Parasitäre Krankheiten; \\
mainDE:Milbeninfektionen;kat:Parasitäre Krankheiten; \\
mainDE:Protozoeninfektion;kat:Parasitäre Krankheiten; \\
mainDE:Giardiasis;kat:Protozoeninfektion; \\
...
```

Abbildung 3.20: Upload-File zum Ausbau der Ontologie mit einer Taxonomie von Krankheiten

Erweiterung der linguistischen Repräsentation

Eine Anwendung des Upload-Files ist die Erweiterung der linguistischen Repräsentation vorhandener Einträge. Dies ist besonders nützlich, wenn die Einträge semiautomatisch erzeugt worden sind, ergänzende Daten für die Attributrepräsentation aber erst nachträglich verfügbar werden, etwa weil sie zwischenzeitlich übersetzt oder aus anderen Quellen gewonnen wurden. Damit wird die Multilingualität der Ressource stark unterstützt und die Zusammenführung unterschiedlicher Phasen der Wissensakquisition (der Modellierungsdaten, der linguistischen Repräsentation, usw.) ermöglicht.

Abb. 3.21 zeigt eine Beispieldatei, mit der die linguistische Repräsentation der Einträge der Krankheitstaxonomie (vgl. Abb. 3.20) um zusätzliche Varianten in verschiedenen Sprachen ergänzt werden kann. Die Daten sind als typisierte Felder kodiert, wobei der Schlüssel durch das Feld `mainDE` und die zusätzlichen Varianten auf Englisch und Latein in Feldern von Typ `en` bzw. `lt` definiert werden. Auf diese Felder wird mit Hilfe von Multivariablen zugegriffen, die zur Definition von optionalen, ergänzenden Attributen benutzt werden. Der Zugriff auf die Einträge erfolgt über eine Query-Komponente, die über den Schlüssel `mainDE` läuft und der Vereinbarkeit mit der Grammatik in Abb. 3.2 halber mit der Komponente `[Topnode]` zu einer Summe ergänzt wurde. Der Identifikator des Topnodes fällt bei der Normalisierung weg, sodass letzten Endes der Abgleich über den ID-String des Krankheitseintrags erfolgt. Der erweiterte ID-String ist hier nicht kodierend und dient nur

als Abfragemechanismus. In der endgültigen Syntax kann auf die Komponente [Topnode] verzichtet werden.

In dieser Art von Sitzung spielen Konfliktbehandlung und Datenexport eine untergeordnete Rolle.

```
\begin{upload} \begin{program} \begin{entries}

        (_{Krankheit}[#mainDE]&[Topnode])

        \begin{names}
        Krankheit.name.de.nomen = "#mainDE"
        Krankheit.variante.de.nomen =? "#*de"
        Krankheit.variante.en.nomen =? "#*en"
        Krankheit.variante.lt.nomen =? "#*lt"
        \end{names}
\end{entries}
\end{program}

\begin{data}{mainDE|en|de|lt}
mainDE:Infektionskrankheiten;en:infectious disease;en:infections;\\
mainDE:Parasitäre Infektion;de:Parasiteninfektion;en:parasitic infections;\\
mainDE:Wurminfektionen;en:worm infections;en:worm infestations;\\
mainDE:Milbeninfektionen;en:mite infections;\\
mainDE:Protozoeninfektion;de:Infektionen durch Protozoen;en:protozoan infections;\\
mainDE:Giardiasis;lt:Lambliasis;en:giardiasis;en:lambliasis;\\
...
```

Abbildung 3.21: Template zur Erweiterung der linguistischen Repräsentation

Herausnehmen bestehender Einträge und Adaption des Netzes

In manchen Situationen kann das Löschen der mit Hilfe eines Upload-Files erzeugten Einträge sinnvoll sein. Das tritt insbesondere ein, wenn die Ressource zur Erschließung einer Textsammlung eingesetzt wird, die eine geschlossene Wissensdomäne darstellt (vgl. Abschnitt *Pflege und Maintenance bzw. Adaption* in Kap. 2, S. 54 ff). Wird eine Kopie des für die Textsammlung relevanten Teils der Ressource in eine separate Datenbank ausgelagert, um diese anschließend anzupassen und weiter zu entwickeln (sog. Adaption) und trägt ein Upload-File nicht zur Abdeckung der anvisierten Textsammlung bei, so empfiehlt sich, zur Vermeidung von Ambiguitäten die entsprechenden Einträge wieder heraus zu nehmen. Beispielsweise stellen die Namen einer Reihe deutscher Gemeinden Homonyme anderer Entitäten aus unterschiedlichen Wissensbereichen dar, wie etwa *Hornbach* (eine Firma) oder *Nußbaum* (eine Baumart). Ist der Wissensbereich *Geographie Deutschlands* bei der

betrachteten Textsammlung nicht relevant, so können durch das Löschen der Gemeinde-Einträge Ambiguitäten vermieden und der relevante Teil der Ressource genauer ermittelt werden. In dieser Hinsicht kann u. U. ein Upload-File einen Beitrag zur Adaption der Ressource leisten. Ähnlich ist dies auch der Fall, wenn bei der Entwicklung der Ressource nachträglich bemerkt wird, dass bestimmte Bereiche zu feingranular ausgebaut wurden und zurückgenommen werden sollten.

Das Herausnehmen erfolgt zunächst durch das Hochladen und Auswerten des entsprechenden Upload-Files. Anschließend können einzelne, als vorhanden identifizierte generierte Einträge ausgewählt und mit einem entsprechenden Knopf im Client gelöscht werden.

Demnach kann ein Upload-File als eine Art ontologisches Modul aufgefasst werden, das nach Bedarf eingespielt oder zurückgenommen werden kann.

Modularisierung vorhandener Upload-Files

Der Einsatz der Exportfunktion zur Erstellung von Differenzdateien wurde bereits angesprochen. Diese Funktion kann dafür verwendet werden, verschiedene Bereiche aus vorhandenen Daten auszuwählen und separat mit geeigneten Schemata abzuspeichern. Beispielsweise können Zeilen einer Tabelle, die eine bestimmte Klasse von Einträgen darstellen und ähnlich kodiert werden, ausgewählt, mittels der Exportfunktion aus der Datei ausgelagert und mit einem entsprechenden Template versehen werden. Wird gleichermaßen festgestellt, dass bestimmte Variablenbelegungen sich sinnvoll nur händisch kodieren lassen, so können diese von den leicht zu integrierenden Daten in eine andere Datei separiert werden. Die Exportfunktion kann ebenfalls eingesetzt werden, bestimmte Belegungen aus einer XML-Datei zu extrahieren und als Tabelle abzuspeichern. Darauf kann anschließend ein Template angewendet werden, das auf XPath-Ausdrücke verzichtet und dadurch leichter zu lesen ist.

3.5 Schlussfolgerungen und Ausblick

In diesem Kapitel wurde eine spezielle Sprache, die Sprache der EFGT-Netz-Schemata oder Templatesprache, entwickelt, mit der sich vorhandene Daten zu EFGT-Netz-Einträgen transformieren lassen. Dabei wurden verschiedene Datenmodelle berücksichtigt, nämlich typisierte Felder, Tabellen und XML, sodass ein breites Spektrum an Daten verarbeitet werden kann. Der eingebaute Abgleichmechanismus erlaubt bei der Generierung neue Einträge von bereits bestehenden zu unterscheiden sowie Konflikte zu erkennen. Insgesamt wird damit eine kontrollierte Integration von Daten in ein bestehendes EFGT-Netz ermöglicht, weswegen die Templatesprache auch als Integrationssprache bezeichnet wird.

Auf der Grundlage der Templatesprache wurde mit dem Upload-Tool ein System entwickelt, das Ontologieentwicklern die halbautomatische Integration von Daten in ein bestehendes EFGT-Netz ermöglicht. Mit dem halbautomatischen Ansatz wurde verfolgt, trotz der automatischen Generierung neuer Einträge die Qualitätssicherung des Integrationsprozesses durch menschliche Ontologieentwickler zu ermöglichen. Als wesentliche Grundlage hierfür dient eine speziell entwickelte Visualisierung der generierten Einträge und

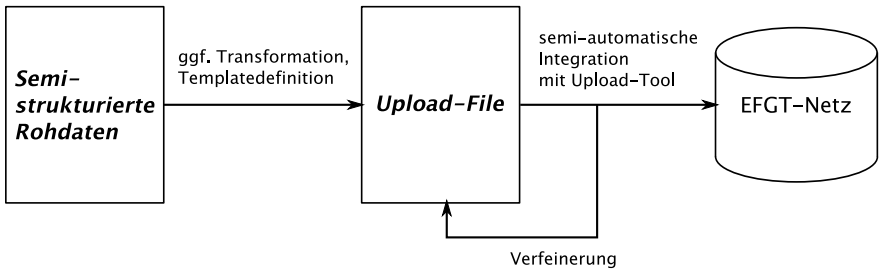


Abbildung 3.22: Gesamter Ablauf bei der Integration semistrukturierter Daten mit Hilfe des Upload-Tools

der Abgleichergebnisse. Mit Hilfe dieser Visualisierung werden eine möglichst effiziente Überprüfung der Ergebnisse und eine effektive Steuerung des Integrationsprozesses miteinander verbunden.

Im Lebenszyklus eines EFGT-Netzes trägt das Upload-Tool wesentlich zu dessen systematischem Aufbau und Population, d.h. zur Phase der Entwicklung, bei. Dies ist wegen der hohen Anzahl der durchzuführenden Einträge besonders relevant für eine linguistische, lexikalische Ontologie (vgl. Kap 1, Abschnitt 1.4) wie das CoGE-Netz, das eine breit angelegte Wissensdomäne anvisiert und die Erfassung eines Teils der natürlichen Sprache anstrebt (s. Kap. 2, Abschnitt 2.2.1). Die Tatsache, dass das CoGE-Netz mit Hilfe des Upload-Tools um Konzepte in der Größenordnung von 10^5 – in der Mehrheit Personen und geographische Begriffe – erweitert werden konnte, verdeutlicht die grundlegende Relevanz des Upload-Tools für die effiziente Entwicklung von EFGT-Netzen.

Der gesamte Ablauf bei der Integration semistrukturierter Daten in EFGT-Netze mit Hilfe des Upload-Tools ist in Abb. 3.22 schematisch dargestellt. Die allgemeine Aufgabe der Extraktion und ggf. Transformation relevanter Daten wurde bewusst als separate, vorgeschaltete Phase der Wissensakquisition betrachtet. Mit den in der Templatesprache berücksichtigten Datenmodellen werden die Anforderungen an das Format, in dem die Ergebnisse dieser Phase vorliegen müssen, möglichst gering gehalten.

Das Upload-Tool ermöglicht den Abgleich der Ontologie mit Daten, die unter einem bestimmten semantischen Gesichtspunkt – z.B. als semantische Klasse oder taxonomische Struktur – in einem Upload-File zusammengetragen wurden. Somit kann ein fokussierter Blick auf die Ontologie geworfen werden, um ihren Zustand in Bezug auf diesen semantischen Aspekt hin gezielt zu überprüfen sowie sie ggf. zu ergänzen. Der Abgleichmechanismus und die bereitgestellten Funktionen ermöglichen, komplizierte Einträge, die etwa eine ausgefallene Modellierung benötigen, zu finden und separat zu behandeln. Diese Funktionalität und der Effekt einer fokussierten Inspektion der Ontologie erlauben dem Upload-Tool ebenfalls, eine wesentliche Rolle in der Phase der Maintenance zu spielen.

Ein Upload-File lässt sich als semantisches Modul betrachten, das der Ontologie hinzugefügt, dank der im Upload-Tool bereitgestellten Funktionen aber auch entnommen werden kann. Letzteres lässt sich als Beitrag zur Adaption an eine bestimmte Domäne betrachten.

In diesem Zusammenhang wäre interessant, Methoden zu entwicklen, die das Synthetisieren eines neuen EFGT-Netzes aus mehreren vorhandenen Upload-Files erlauben. Hierfür bietet die hier entwickelte Integrationssprache einen guten Ausgangspunkt.

Die Schematisierung von ID-Strings, wie sie in Templates für verschiedene semantische Bereiche erfolgen kann, trägt dazu bei, die Konzeptualisierung unterschiedlicher Arten von Einträgen transparent zu machen. In dieser Hinsicht ist es denkbar, dass die Zusammenstellung einer Bibliothek von Templates für ein spezifisches EFGT-Netz dazu beitragen kann, die Entwicklung zu dokumentieren und neue Datenquellen leichter zu integrieren. Beispielsweise könnte das Führen von Statistiken über die Verwendung verschiedener Schemata dazu dienen, automatisch Vorschläge zu generieren. Eine Bibliothek von Templates könnte eine Grundlage darstellen, die Entwicklung in kollaborativen Szenarien durch naive Benutzer zu steuern.

Unabhängig vom Einsatz innerhalb des Upload-Tools kann die entwickelte Integrationssprache als wesentlicher Beitrag zur Automatisierung verschiedener Prozesse zum Ontologieaufbau betrachtet werden. Beispielsweise lässt sich ein System implementieren, das regelmäßig bestimmte Internetquellen zum aktuellen Kinoprogramm besucht, die entsprechenden Seiten mit den Filmlisten mit Hilfe von Wrappern in ein Upload-Tool transformiert, um die neuen Filme dann anschließend in ein EFGT-Netz zu integrieren.

Neben den im Laufe des Kapitels angesprochenen unmittelbaren Weiterentwicklungen des Upload-Tools sind weitere Verbesserungen denkbar. Dies betrifft insbesondere die Visualisierung von XML-Daten im Client, in der bisher die Anzeige der Instanziierungen des Templates ohne Bezug auf die entsprechenden Stellen in den Originaldaten erfolgt. Die graphische Darstellung der von dem definierten Template betroffenen Datenbereiche sowie ein unmittelbares Feedback auf die verwendeten XPath-Ausdrücke wären hier sinnvolle Erweiterungen, die die Definition von Verweisen zur Erfassung funktionaler Abhängigkeiten in XML-Daten erleichtern würden. Unabhängig davon kann in Betracht gezogen werden, die Templatesprache zu einer Abfragesprache zu erweitern, um etwa unter einem bestimmten semantischen Gesichtspunkt Listen von Konzepten aus einem EFGT-Netz herauszuziehen und diese dann weiter zu verarbeiten.

Kapitel 4

Thematische Suche und Navigation in Dokumentarchiven

4.1 Motivation für den thematischen Archivbrowser

Eine wichtige Erkenntnis, die sich aus der Betrachtung und dem Vergleich verschiedener Methodologien zur Ontologieentwicklung in Kapitel 1 ergibt, lässt sich damit zusammenfassen, dass die frühzeitige Einbindung der Ontologie innerhalb einer Anwendung als Kontrollschritt im Ontologie-Lebenszyklus dient und zur Qualität der entwickelten Ressource wesentlich beitragen kann. Eine Anwendung stellt einen konkreten Kontext dar, in dem die Ontologie getestet wird und der dadurch Vorschläge zu ihrer weiteren Korrektur und Verfeinerung liefern kann. Aus diesem Grund wurde als zweites Ziel dieser Dissertation festgelegt, eine exemplarische Anwendung zu erstellen, in der ein Bezug zwischen einem spezifischen EFGT-Netz und einer bestimmten Dokumentensammlung hergestellt wird (s. S. 58). Hierbei repräsentiert die Dokumentensammlung eine bestimmte Wissensdomäne.

Die in diesem Kapitel zu entwickelnde Anwendung soll ein Feedback über den Entwicklungsstand eines spezifischen EFGT-Netzes liefern. Dieses Feedback ist nach der Integration großer Datenmengen in die Ressource, wie es mit dem Upload-Tool möglich ist, besonders wichtig, weil sich das EFGT-Netz von einzelnen Ontologieentwicklern kaum noch in seiner Gesamtheit überblicken lässt. Insbesondere sollen aus der Interaktion mit der Anwendung Einsichten über

- die Kodierung der Einträge und die sich daraus ergebende Struktur des EFGT-Netzes
- die Abdeckung der Wissensdomäne und die Qualität der linguistischen Repräsentation der Konzepte

gewonnen werden. Diese zwei Elemente, Struktur und Vokabular, stehen im Fokus des Ontologieentwicklungsprozesses und so ist die Möglichkeit der Interaktion mit der Ontologie in einem Anwendungskontext gerade in der Entwicklungsphase wichtig. Dadurch kann die ausgearbeitete Modellierung und deren unmittelbare Auswirkung auf die Anwendung vom

Ontologieentwickler beobachtet werden. Darüber hinaus kann Aufmerksamkeit auf spezifische Aspekte der Wissensdomäne gelenkt werden, die allein aus der abstrakten Betrachtung heraus unentdeckt bleiben würden.

Struktur und Vokabular stellen außerdem die wichtigsten Ebenen dar, die es bei der Evaluation von EFGT-Netzen und Ontologien im allgemeinen zu betrachten gilt (vgl. Abschnitt *Evaluation von Ontologien*, S. 28). Auch wenn eine exemplarische Anwendung zunächst nur ein qualitatives Feedback über den Entwicklungsstand der Ressource liefert und keine quantitative Evaluation ersetzen kann, kann sie dazu dienen, erst einmal die Aspekte zu identifizieren, die mit Hilfe quantitativer Methoden evaluiert werden sollen. So kann die Einbindung der Ontologie in eine Anwendung als eine ergänzende Maßnahme zur automatisierten, quantitativen Evaluation der Ressoruce verstanden werden.

Die Ausgangsidee für die in diesem Kapitel vorgestellte Anwendung besteht darin, auf der Grundlage des EFGT-Netzes ein thematisches Inhaltsverzeichnis aufzubauen, über das strukturiert auf Dokumente der Sammlung zugegriffen werden kann. Die Einträge und die Struktur dieses Verzeichnisses korrespondiert mit dem Teil des EFGT-Netzes, das die Thematik und den durch die Textsammlung repräsentierten Wissensbereich abdeckt. Wählt man ein Konzept des Verzeichnisses aus, so verschafft die Anwendung Zugang auf die Dokumente, die – in einer näher zu definierenden Form – "relevant" für das Konzept sind. Der Name dieser Anwendung, der *thematische Archivbrowser*, deutet diese Funktionalität an.

Dadurch, dass der Benutzer zur Sichtung der für ein spezifisches Konzept relevanten Dokumente im thematischen Inhaltsverzeichnis navigieren muss, was dem Navigieren im relevaten Teil des EFGT-Netzes gleicht, kann ein Eindruck über die Struktur der Ressource und deren Entwicklungsstand gewonnen werden. Die Zuordnung von Dokumenten zu einem bestimmten Konzept, die auf Grundlage der im Dokument erkannten Konzepte und deren Beziehungen zueinander im EFGT-Netz erfolgt, liefert implizit einen weiteres Feedback über die sich aus der Kodierung ergebende Netzstruktur. Durch die Sichtung von Dokumenten, die für ein Konzept relevant sind, und der darin vorkommenden Begriffe, entsteht für den Benutzer ein Bild von der Abdeckung des jeweiligen Wissensbereichs.

Bei der Übertragung des EFGT-Netzes als thematisches Inhaltsverzeichnis in die Anwendung wird die ursprüngliche Struktur weitgehend erhalten. Zudem basiert die Berechnung der Abbildung von Konzepten auf Mengen relevanter Dokumente auf einfachen und dadurch transparenten Algorithmen, die auf verschiedene Möglichkeiten der Optimierung verzichten. Dadurch wird ein unverfälschtes Bild über den Zustand der Ontologie und dessen Folgen für den thematischen Archivbrowser vermittelt, was aus der Sicht des Ontologieentwicklers besonders nützlich ist.

Der thematische Archivbrowser wurde eingesetzt, um die Entwicklung des CoGE-Netzes zu begleiten, das Allgemeinwissen erfassen soll (s. Kap 2). Da als repräsentative Dokumentsammlung hierfür die Nachrichtenerstattung in der Internetpräsenz deutscher Presseverlage zugrunde gelegt wird, ist der Vergleich der thematischen Navigation mit den Techniken, die von Online-Medien verwendet werden, um den Zugriff auf ihre Inhalte zu strukturieren, interessant – nicht zuletzt deshalb, weil der Archivbrowser ein Eigenleben als Anwendung hat und als selbständiges Produkt betrachtet werden kann.

4.2 Online-Navigation in Pressearchiven

Seitdem das Internet zu einem öffentlichen Kommunikationsmedium geworden ist, konkurrieren klassische Printmedienunternehmen, insbesondere im Bereich der Presse und Nachrichtenerstattung, nicht nur mit ihren gedruckten Auflagen miteinander sondern immer stärker über ihre Onlinepräsenz. Im neuen Medium begegnet Verlagen ein breites Spektrum von Anbietern konkurrierender Inhalte, zu denen auch neue Formen der Berichterstattung und Meinungsbildung wie Blogs, Diskussionsforen, kollaborativ aufgebaute Enzyklopädien, usw. zählen. Gegenüber diesen Anbietern gilt es, die bisher als Printmedium innegehabte Position zu behaupten. Nach Jahren rückläufiger gedruckter Auflagen hat das Internet den Printmedien den Rang als Marktplatz für das Geschäft mit Werbung – ein wichtiges finanzielles Standbein von Zeitschriftenverlagen – den Rang abgelaufen. Im neuen Medium gilt es, ausfallende Werbeeinnahmen auszugleichen und aus einem potenziell größeren Leserkreis Nutzen zu schöpfen.

Vor diesem Hintergrund ist das primäre Ziel, über den Online-Auftritt eine hohe Nutzerbindung zu erreichen und neue Nutzerkreise zu erschließen. Eine mögliche Strategie, um Internetnutzer zu bündeln und als regelmäßige Besucher zu gewinnen, besteht darin, neue Inhalte und Dienste in die Internetpräsenz einzubinden. Eine komplementäre Strategie versucht, die im Pressebereich traditionell angebotenen Inhalte neu zu präsentieren und aufzubereiten. Insbesondere ist es attraktiv, die Masse an über Jahren angesammelten Informationen in Form von Archiven zu verwerten, indem sie über das Internet einem breiterem Publikum zugänglich gemacht werden. Indem der Presseverlag sein Nachrichtenarchiv oder spezielle Dossiers anbietet, profiliert er sich nicht nur als Lieferant von Aktualität sondern auch von Hintergrundinformationen mit dem Ziel, Online-Enyklopädien und Nachschlagewerken einen Teil ihrer Nutzer streitig zu machen. Bei der Vermarktung von Archiven spielen effektive Navigationsmechanismen in der Internetpräsenz eine zentrale Rolle, mit denen erreicht werden soll, dass möglichst viel der angebotenen Information vom Benutzer gesichtet wird. Die in diesem Kapitel vorgestellte thematische Navigation wird später (Abschnitt 4.5, S. 130) mit gängig eingesetzten Mechanismen verglichen. Bevor verschiedene Realisierungen von Navigation vorgestellt werden, sei an dieser Stelle der Vollständigkeit halber auf die erste Strategie eingegangen.

Einbindung alternativer Dienste und Inhalte. Die Einbindung alternativer Dienste und Inhalte soll aus verschiedenen Gründen zu einer hohen Bindung der Nutzer führen:

- Neben der geschriebenen Nachrichtenerstattung als Quelle von Aktualität ergänzen zusätzliche Inhalte, oft von Drittanbietern, das eigene Angebot um Hintergrundinformationen. Beispielsweise präsentiert die Volltextsuche auf der Online-Ausgabe des Spiegel – unter den Namen SPIEGEL WISSEN (vgl. SPIEGEL ONLINE GmbH, 2008) vermarktet – als Ergebnisse einer Suche sowohl Artikel aus dem eigenen Archiv als auch einschlägige Inhalte von Drittanbietern, etwa Artikel aus bekannten Nachschlagewerken, aus Zeitschriften derselben Unternehmensgruppe sowie Statistiken und Studien. Damit wird jenseits von Aktualität auf eine Profilierung als allge-

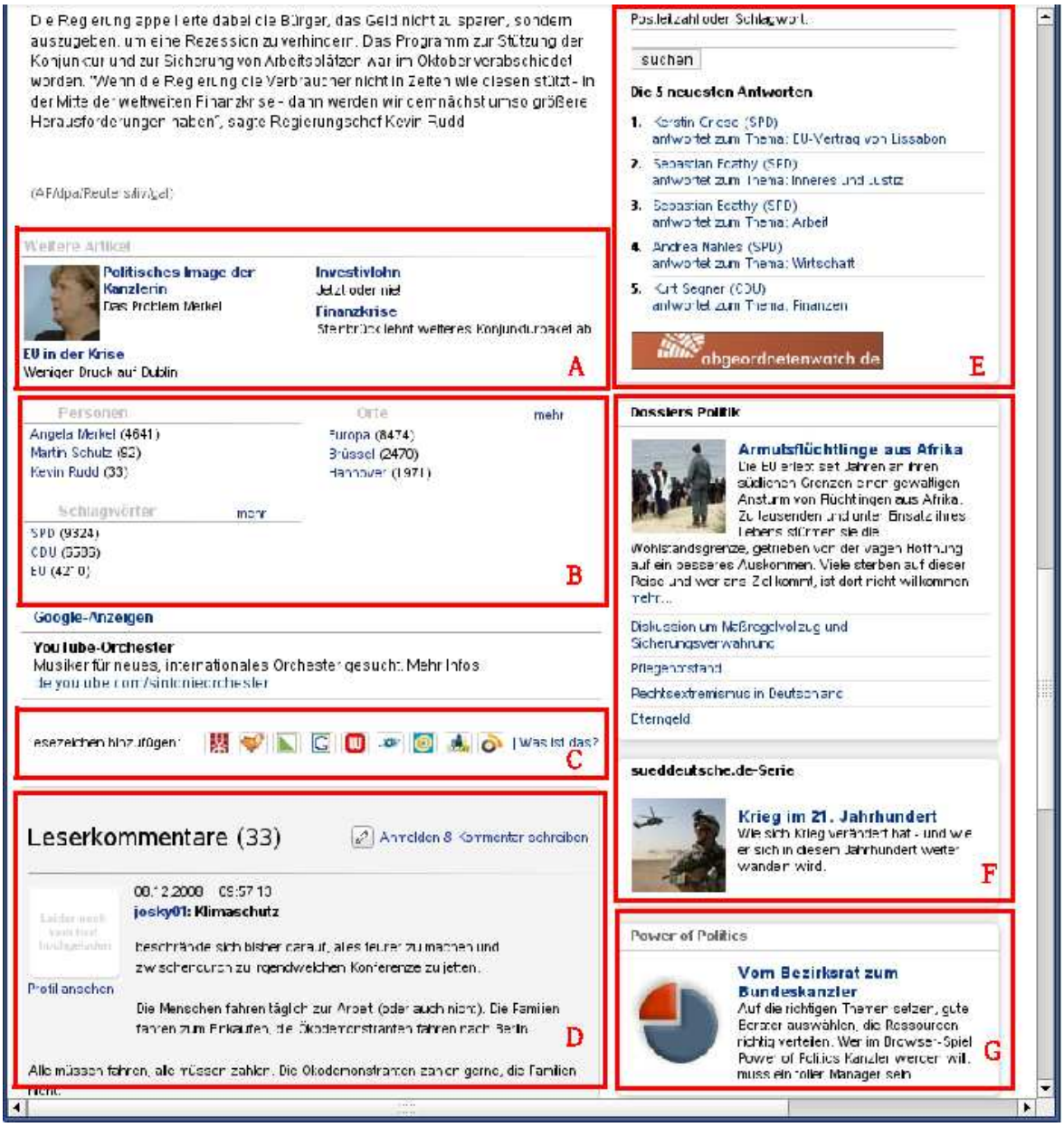


Abbildung 4.1: Navigationselemente und zusätzliche Dienste auf *sueddeutsche.de* (unterer Bereich der Seitenaufmachung für einen Artikel zum Thema Politik). Navigationselemente für den Zugang auf eigene Inhalte: Verweise auf verwandte Artikel (A) und auf Dossiers (F); Entitätenarten-basierte Navigation (B). Zusätzliche Dienste, die über die klassische Nachrichtenerstattung hinausgehen: Externe Anbieter von *social bookmarking*-Diensten und Nachrichtenaustausch- und Bewertungsbörsen (C). Logotypen, von li. nach re.: Mr. Wong; Yigg; Linkarena; Google; Webnews; Folkd; Oneview; Lufee; Wikio. Dienst zum Meinungsaustausch mit anderen Lesern (D). Thematisch einschlägige Dienste: Anfragedienst an Abgeordente (E); politisches Spiel (G).

meine Informationsquelle abgezielt. Die zusätzlichen Inhalte stellen zudem oft eine alternative Art der Informationsbereitstellung dar. So gehören inzwischen Multimedia-Inhalte wie Videos, interaktive Graphiken oder Bilderstrecken sowie News-Ticker, Blogs usw. zum üblichen Bild, mit dem sich Zeitungen im Internet präsentieren (s. Abb. 4.1 und 4.2), sodass die Präferenzen bestimmter Nutzergruppen berücksichtigt werden.

- Dienste, die über die traditionell angebotenen wie Wetter oder Veranstaltungskalender hinausgehen, sollen dazu beitragen, den Internetauftritt des Presseanbieters als Anlaufstelle in verschiedenen Lebenssituationen zu betrachten. So können beispielsweise auf der Internetpräsenz der Süddeutschen Zeitung (vgl. sueddeutsche.de GmbH, 2008) unter der Rubrik *Infothek* verschiedene Dienste mit Titeln wie Telefontarife, Software Portal, Hartz IV-Rechner, usw. genutzt werden. Andere Dienste werden in Abhängigkeit des Verhaltens des Nutzers angeboten (s. Abb. 4.1).

- Indem der Besucher die Möglichkeit erhält, Nachrichten zu bewerten, zu kommentieren und zu teilen, soll er involviert werden und die emotionale Bindung an die Website gesteigert werden, die sich als Diskussionsplattform aktueller Themen präsentiert (s. Abb. 4.1).

- Zur Distribution und Steigerung der Sichtbarkeit der eigenen Inhalte wird auf Mechanismen wie RSS-Feeds-Services oder *social Bookmarking*-Dienste zurückgegriffen. Eine Erreichbarkeit der Inhalte über Suchmaschinen wird nicht aktiv unterstützt, da diese als zentraler Eingang zum Internet eine Art Vormachtstellung besitzen und gerade bezüglich Werbung über Vermarktungsmöglichkeiten verfügen, die stark in Konkurrenz mit denen der Presseanbieter treten und sich deren Kontrolle entziehen.

Navigationsmechanismen zur Aufbereitung von Presseausgaben und Archiven. Im Internet können Presseinhalte in einer Form präsentiert und aufbereitet werden, die über die Möglichkeiten der gedruckten Ausgabe hinausgehen. Das trifft insbesondere auf Pressearchive zu und eröffnet die Möglichkeit, diese für ein breiteres Publikum zu vermarkten.

In diesem Zusammenhang spielen effektive Formen der Navigation und des Zugriffs auf Inhalte eine zentrale Rolle. Diese sollen einerseits dem Benutzer bei der Informationssuche in der Masse der Dokumente eines Archivs helfen, sich zu orientieren, und ihn unterstützen und sind eine wichtige Voraussetzung dafür, dass er das Angebot überhaupt sinnvoll nutzen kann. Andererseits sollen Navigationsmechanismen dazu animieren, in den Inhalten zu stöbern und so längere Verweilzeiten oder eine höhere Anzahl von Pageimpressions – wichtige Kennzahlen im Geschäft der Internet-Werbung – zu erreichen, die zu einer profitablen Schaltung von Werbeinhalten führen.

Anhand von Beispielen wird im folgenden auf aktuell verwendete Navigationselemente eingegangen, mit denen Printmedien das Ziel verfolgen, dass ein Nutzer auf deren Internetpräsenz möglichst viele Inhalte sichtet. Ein allgemeiner Überblick über verschiedene

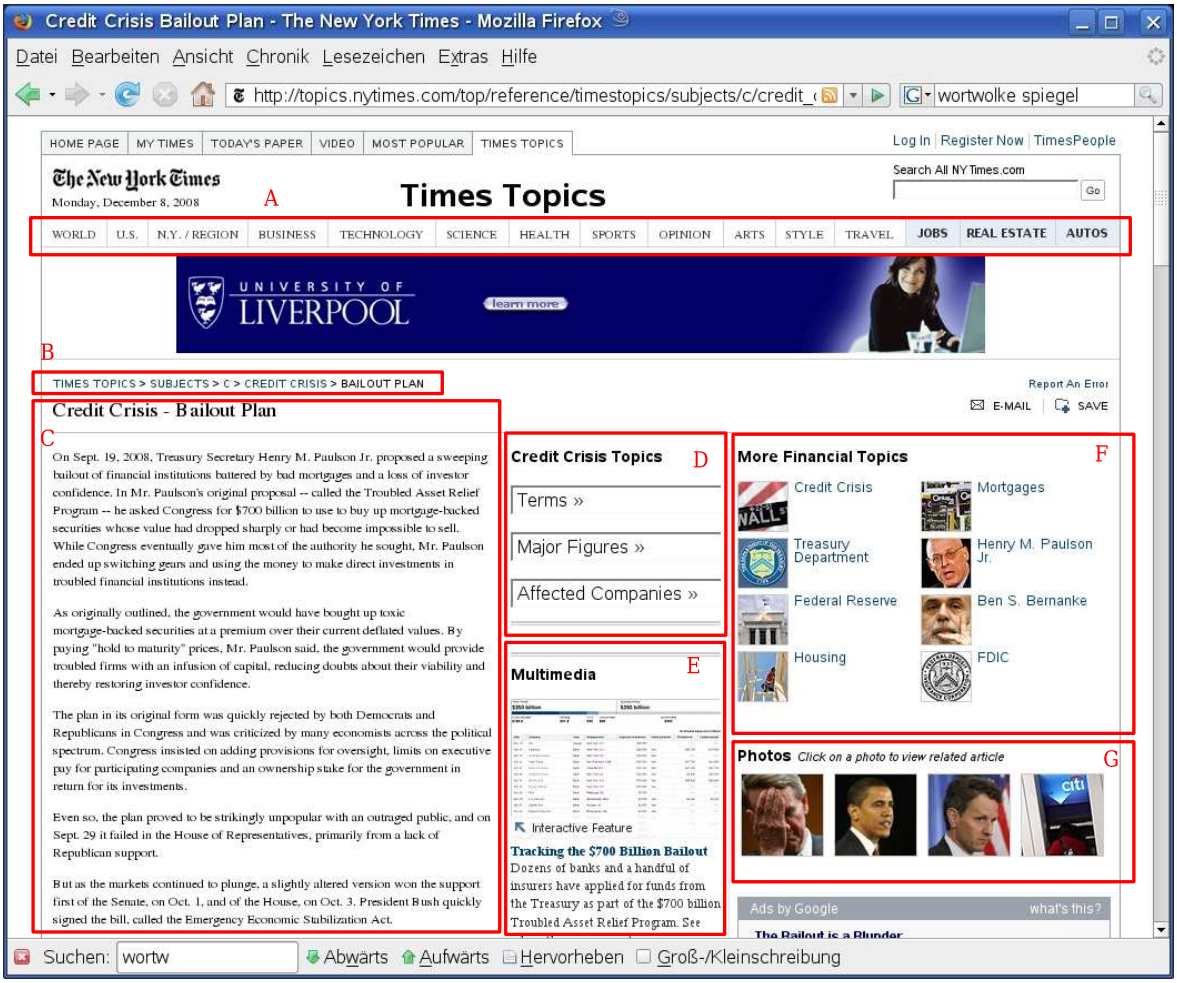


Abbildung 4.2: Thematische Kategorien (*topics*) bei der *New York Times*. Ein *topic* kann für einen Begriff, ein Ereignis, eine Person, eine Institution, usw. stehen, wofür jeweils Hintergrundinformationen bereitgestellt werden (C). Topics werden zur Beschlagwortung von Inhalten verwendet, etwa Artikeln, Multimediainhalten (E) oder Fotos (G). Topics werden auch in Beziehung zueinander gestellt, wobei hier unterschiedliche Typen berücksichtigt werden (D). Dadurch wird der Leser auf wichtige Zusammenhänge aufmerksam gemacht und zum Stöbern animiert. Mit *topics* lassen sich ebenfalls Navigationselemente wie verwandte Kategorien (Bereich F) umsetzen. Weitere Navigationsmechanismen auf dem Bild sind der Reiter zur globalen Navigation über Rubriken (A) und sogenannte *location bread scrums* (B), die die Position der Seite in der Struktur des Webauftritts verdeutlichen.

Navigationselemente und -ansätze beim Aufbau einer Internetpräsenz findet sich bei Kalbach (vgl. Kalbach, 2008).

Als *globale Navigation*, die eine Orientierung in der gesamten Webpräsenz ermöglicht, dient meistens ein System von Rubriken und ggf. zugehörigen Unterkategorien, die den im Pressebereich traditionell verwendeten Rubriken entsprechen. Dadurch wird eine Or-

Abbildung 4.3: Wortwolke als Themen des Tages bei Spiegel ONLINE. Die angezeigten Begriffe stellen Querverweise auf Artikel dar, die den jeweiligen Begriff enthalten. Die Schriftgröße deutet die Relevanz des Begriffs im Vergleich zu anderen Themen des Tages an. Diese Relevanz kann sowohl statistisch ermittelt als auch manuell gesetzt werden.

ganisation der gesamten Webpräsenz in einer relativ flachen Struktur von Bereichen erreicht. Klassisch ist ebenfalls die Bereitstellung einer Funktion zur Stichwortsuche. Diese ermöglicht einen effizienten Zugang auf Inhalte insbesondere in den Fällen, in denen von einem präzisen, vorab geklärten Informationsbedarf ausgegangen werden kann. Beispiele für die Darstellung dieser beiden Mechanismen sind in Abb. 4.2 und Abb. 4.5 zu sehen.

Ergänzend zur globalen Navigation werden verschiedene Mechanismen eingesetzt, die auf benannten Entitäten und feststehenden Begriffen aufbauen, die selbst in den Nachrichtentexten vorkommen:

- Wortwolken oder Tagclouds (Abb. 4.3) vermitteln einen graphischen Eindruck von der Häufigkeit oder der Relevanz wichtiger Begriffe im Archiv und ermöglichen den Zugang auf Dokumente, die die dargestellten Ausdrücke enthalten. Die in einer Wortwolke angezeigten Begriffe werden meistens auf der Grundlage der Texte im Archiv automatisch ermittelt, können aber auch in manchen Fällen ein System händisch gepflegter Schlagwörter widerspiegeln.

- Eingebettet in den Text können Ausdrücke eines kontrollierten Vokabulars als Querverweise zu Hintergrundinformationen oder anderen Nachrichten verwendet werden. Damit wird eine *assoziative Navigation* ermöglicht (Kalbach, 2008, S. 93), mit der wichtige Querverbindungen über Hierarchieebenen der Website hinweg ermöglicht werden. Abb. 4.2 zeigt die Zielseite eines Querverweises für das Wort *bailout* (dt.: Schuldenübernahme durch Dritte) in einem Artikel der New York Times über die aktuelle Finanzkrise. In diesem Fall enthält die Zielseite einen Hintergrundartikel über eine Schuldübernahmeplan seitens des Staates in den USA, Verweise auf Multimediaangebote und verwandte Themen sowie einschlägige Werbeinhalte.

- In einem weiteren Ansatz werden die einzelnen Dokumente des Archivs mit darin vorkommenden Entitäten und wichtigen Ausdrücken beschlagwortet, die entweder automatisch erkannt wurden oder Teil eines kontrollierten Vokabulars sind. Zur Navigation dienen dann Verweise auf andere Dokumente, die Schlagwörter mit dem angezeigten Dokument teilen. Ein Beispiel ist in Abb. 4.1 zu sehen (Bereich B), wobei die Schlagwörter teilweise nach Entitätentypen gruppiert sind.

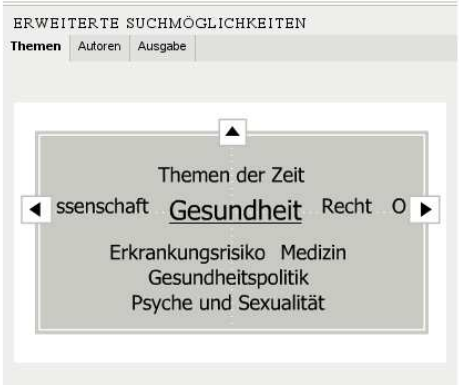


Abbildung 4.4: Themenbrowser als Zugangsfunktion auf das *Zeit online*-Archiv. Dargestellt ist der Ausschnitt der Themenhierarchie für die Kategorie *Gesundheit*, die unmittelbar unterhalb der Wurzel *Themen der Zeit* liegt. Unterhalb befinden sich Verfeinerungen des Themas *Gesundheit*; links und rechts Schwesterkategorien. Die angezeigten Kategorien stellen Querverweise auf dynamisch erstellte Dossiers zum jeweiligen Thema dar, wobei die Artikel hierfür mit entsprechenden Suchbegriffen beschlagwortet wurden. Über die Reiter lassen sich die Artikel nach einem bestimmten Autor filtern (*Autoren*) sowie das Erscheinungsdatum eingrenzen (*Ausgabe*).

Ein kontrolliertes Vokabular aus relevanten benannten Entitäten und feststehenden Begriffen kann nicht nur zur Beschlagwortung von Dokumenten eingesetzt werden, sondern gleichzeitig als Grundlage für ein leicht strukturiertes System semantischer Kategorien oder Themenbereiche (engl.: *topics*) dienen, mit dem komplementär zum globalen Rubrikensystem die Navigation in der Website ermöglicht wird. So bietet die New York Times eine Liste von über 13.000 alphabetisch geordneten *topics* an, die zu den entsprechend beschlagworteten Artikeln führt. Oft bestehen Beziehungen zwischen verschiedenen Kategorien zueinander. Die üblichste Beziehungsart ist die thematische Verwandtheit, die zur Realisierung assoziativer Navigationselemente eingesetzt wird. Bereich F in Abb. 4.2 zeigt ein Beispiel hiervon. Auf derselben Abbildung lässt sich auch beobachten, dass spezifische Beziehungsarten zwischen *topics* und anderen Elementen des kontrollierten Vokabulars zur Navigation eingesetzt werden. So stellen die aufklappbaren Menüs im Bereich D jeweils eine Relation des angezeigten *topic* mit wichtigen Begriffen (*Terms*), Personen (*Major Figures*) und Unternehmen (*Affected Companies*) dar und führen zu den entsprechenden Inhalten.

Auf der Grundlage der Beschlagwortung lassen sich ebenfalls Dossiers zu einzelnen Themenbereichen dynamisch erstellen und somit das Angebot der manuell zusammengestellten Dossiers erweitern. Dieser Ansatz wird bei *Zeit Online* verfolgt, wobei die Kategorien in einem Baum von Themen organisiert wurden, in dem mit Hilfe des sogenannten Themenbrowsers navigiert werden kann (Abb. 4.4).

Ausgehend von unterschiedlichen Typen benannter und thematischer Entitäten lassen sich für ein Archiv verschiedene Arten von Inhaltsverzeichnissen realisieren. So kann bei *ElPaís.com* auf die aktuelle Ausgabe von unterschiedlichen Perspektiven aus zugegrif-

fen werden: Neben einem Verzeichnis, das von den gängigen Rubriken ausgeht, stehen Verzeichnisse jeweils nach geographischen Regionen, nach vorkommenden Personen und Organisationen (Abb.4.5) und nach thematischen Kategorien (Abb. 4.6) zur Verfügung. Ebenfalls lassen sich Auflistungen der Inhalte nach ihrer Art (Videos, Fotos, Graphiken, usw.) aufrufen.

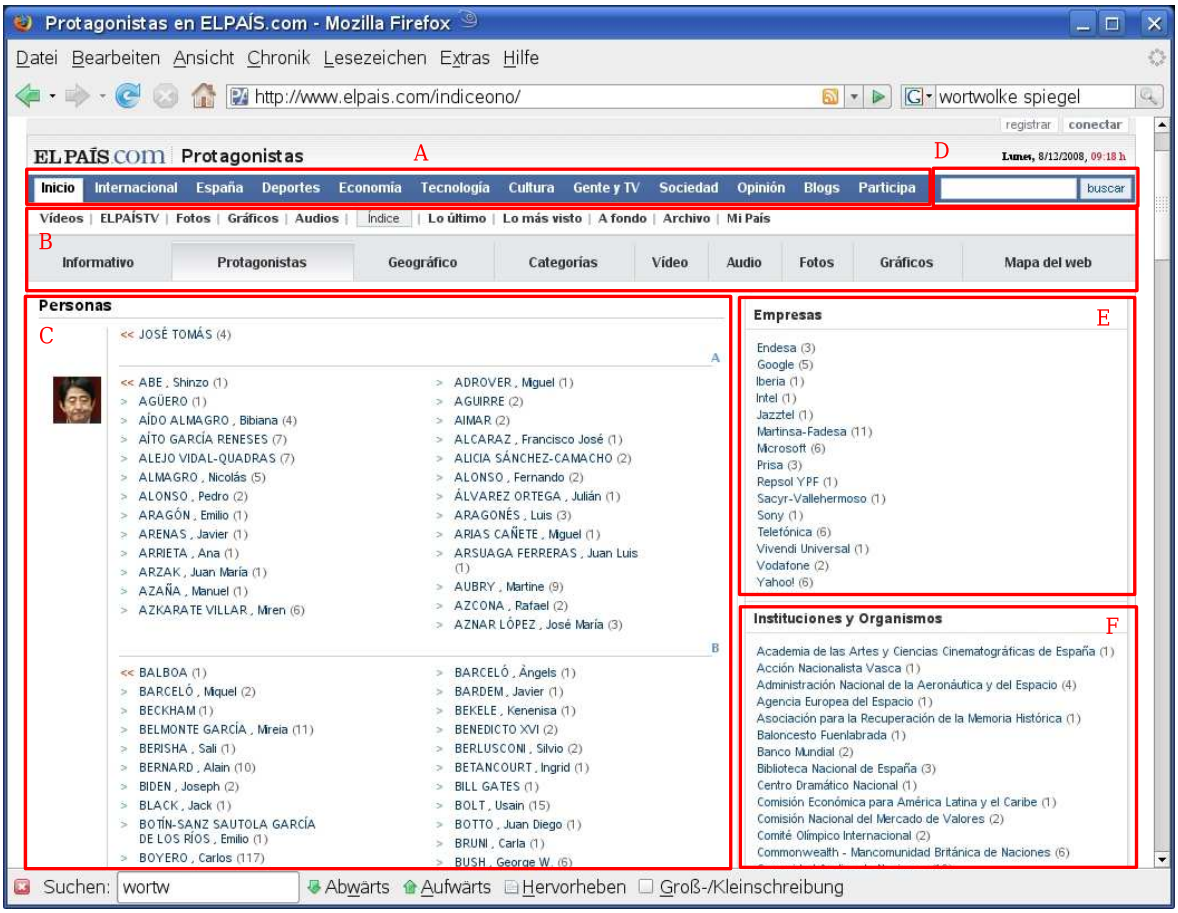


Abbildung 4.5: Personen- und Organisationenverzeichnis für die tägliche Onlineausgabe von *El País*. Dieses spezielle Verzeichnis wurde über den Reiter zur Hauptnavigation (B) aufgeschlagen und zeigt eine alphabetisch geordnete Liste von Personen (C), Unternehmen (E) sowie Organisationen und Einrichtugen (F). Durch das Anklicken einer dieser Entitäten gelangt man zu den Artikeln, in denen diese Entität textuell erwähnt wird. Die Navigation über Rubriken erfolgt mit Reiter A, eine Volltextsuche ist über das Eingabefeld D möglich.

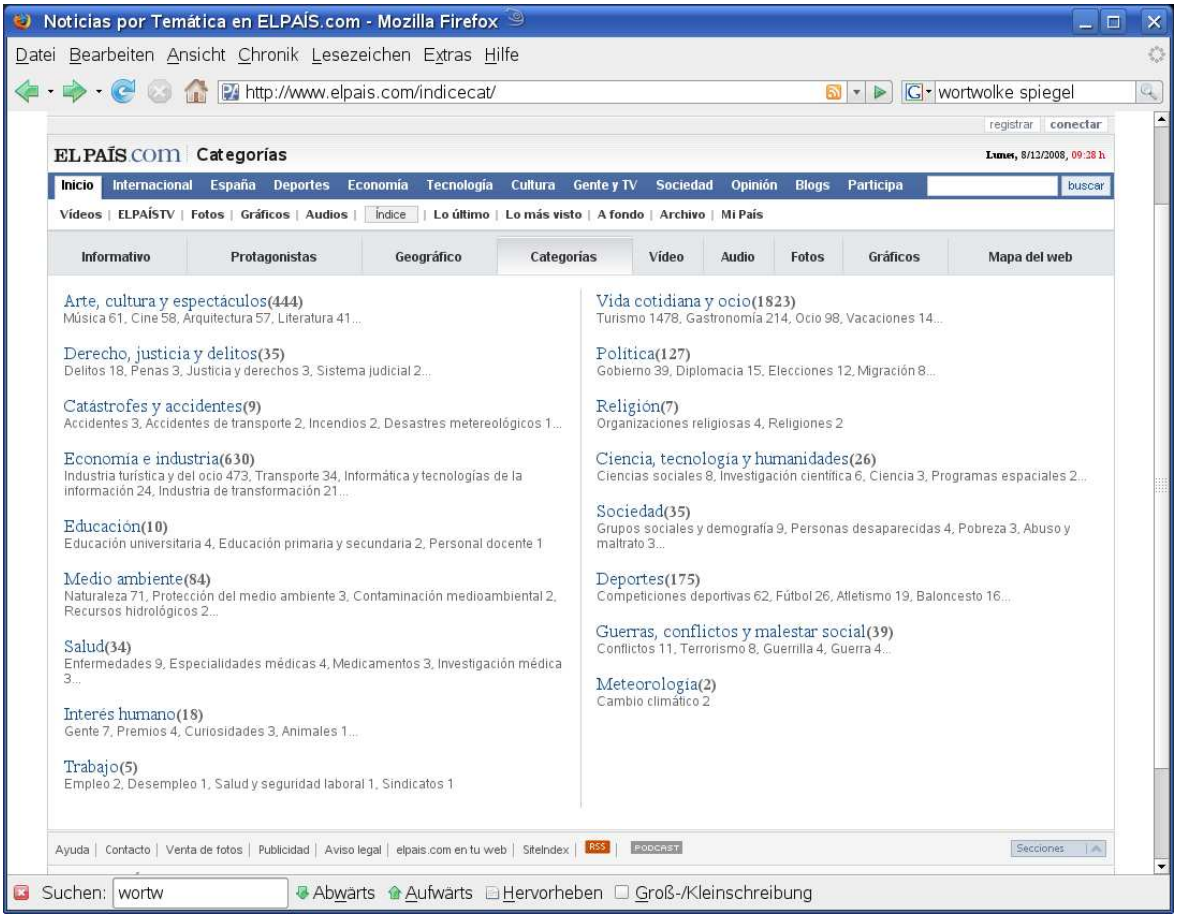


Abbildung 4.6: Thematisches Inhaltsverzeichnis bei *ElPaís.com*. Das dargestellte Verzeichnis entspricht derselben Ausgabe wie in Abb. 4.5 und ermöglicht den Zugang auf Artikel über einen flachen Kategorienbaum, der in seiner Darstellung an Webverzeichnisse erinnert. Die Artikel werden über eine entsprechende Beschlagwortung den Kategorien zugeordnet.

4.3 Der thematische Archivbrowser

Im thematischen Archivbrowser dient das EFGT-Netz als eine Art Inhaltsverzeichnis der Dokumentensammlung. Dem Benutzer steht für die Navigation und Suche im Verzeichnis und der Dokumentsammlung ein Client zur Verfügung, der sich in einem gängigen Webbrowser aufrufen lässt und dessen Funktionalität im folgenden beschrieben wird. Nach der Darstellung der Architektur des thematischen Browsers wird näher auf das Verfahren eingegangen, mit dem das thematische Inhaltsverzeichnis aufgebaut wird und den Konzepten des Netzes Dokumente zugewiesen werden.

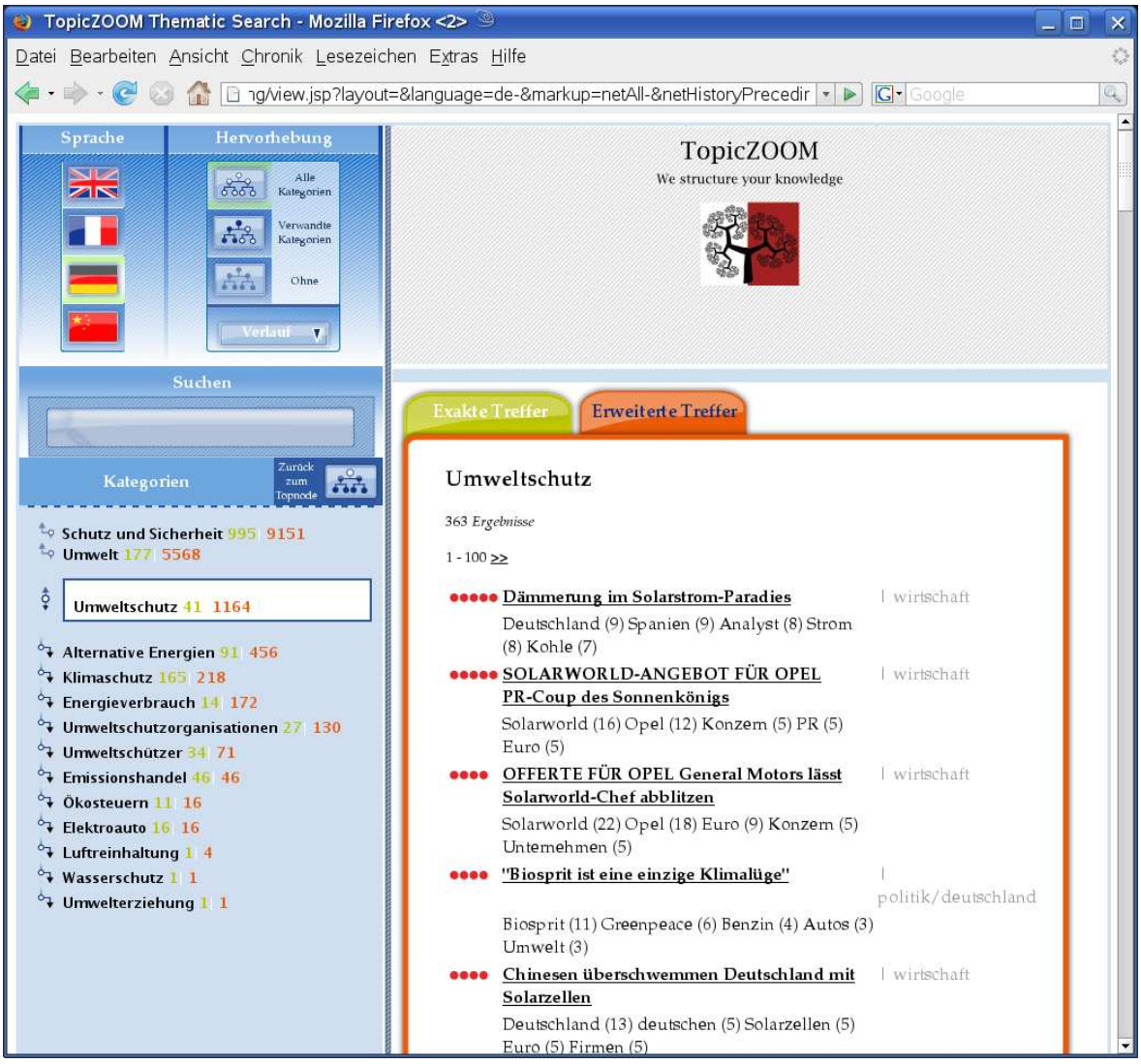


Abbildung 4.7: Die Benutzeroberfläche des thematischen Archivbrowsers. In diesem Fall entspricht der indexierte Korpus einer Sammlung von Zeitungsartikeln.

4.3.1 Thematische Navigation und Suche

Aus der Sicht des Benutzers präsentiert sich der thematische Archivbrowser wie in Abb. 4.7 dargestellt. Die linke Spalte enthält verschiedene Knöpfe zur Steuerung der Anzeige (Bereiche mit der Überschift *Sprache* bzw. *Hervorhebung*), ein Suchfeld (mit der Überschrift *Suche*) sowie die sogenannte *Konzeptumgebung*, die mit *Kategorien* beschriftet ist. In der Konzeptumgebung erscheinen das aktuelle Konzept – *Umweltschutz* in der Abbildung – sowie dessen umittelbare Vor- und Nachfahren im Netz. Die Navigation in der Netzstruktur erfolgt, indem eines der in der Umgebung erscheinenden Konzepte angeklickt wird, das dadurch zum aktuellen Konzept wird. Je nachdem, ob man einen unmittelbaren Vorfahren

Abbildung 4.8: Auto-Vervollständigungsfunktion im Suchfeld des thematischen Archivbrowsers. Nachdem die Anfangsbuchstaben eines Suchbegriffs eingegeben werden, erscheint eine Liste einschlägiger, im EFGT-Netz erfasster linguistischer Varianten, über die sich ein bestimmtes Konzept auswählen lässt.

oder einen Nachfahren auswählt, bewegt man sich nach oben – zum Topnode hin – bzw. nach unten im Netz.

Auf der rechten Seite erscheinen die Ergebnislisten zum aktuellen Konzept. Es werden zwei Modalitäten von Listen relevanter Dokumente angeboten, sogenannte *exakte* und *erweiterte Treffer*, die über den entsprechenden Reiter ausgewählt werden können. Exakte Treffer sind Dokumente, in denen das aktuelle Konzept vorkommt, während bei den erweiterten Treffern nur sichergestellt ist, dass zumindest ein Nachfahre des aktuellen Konzepts im Netz textuell vorhanden ist. Erweiterte Treffer berücksichtigen somit die Struktur des Netzes, um relevante Dokumente für das aktuelle Konzept zu ermitteln.

Die Zahlen neben den Konzepten in der Konzeptumgebung entsprechen jeweils der Anzahl dieser beiden Arten von Treffern für das aktuelle Konzept. Durch Anklicken können sie dazu genutzt werden, das aktuelle Konzept zu wechseln und gleichzeitig die gewählte Art der Trefferliste anzuzeigen. Diese Zahlen stellen die Verteilung der Dokumentensammlung über die in der Konzeptumgebung angezeigten Kategorien dar – eine Art Verteilungsprofil.

Im Suchfeld können die Anfangsbuchstaben eines Suchbegriffes eingegeben werden, wodurch eine Liste von Ausdrücken aufgebaut wird, zu denen der angegebe Prefix vervollständigt werden kann (sog. *Auto-Vervollständigung*, s. Abb. 4.8). Die Ausdrücke entsprechen Vorzugsbezeichnungen und Varianten der linguistischen Repräsentation von in der Datenbank vorhandenen Konzepten, sodass durch Auswahl einer dieser Ausdrücke im Client direkt in die Konzeptumgebung des entsprechenden Konzepts gewechselt werden kann. Ein solches Suchfeld mit Auto-Vervollständigung schafft eine Brücke zu den Erwartungen an ein Suchsystem, die durch verbreitete Verwendung von Internet-Suchmaschinen entstanden sind, und ist besonders nützlich, wenn ein bestimmtes, aus dem Netz bekanntes Konzept gezielt gesucht wird.

In den Ergebnislisten werden die Dokumente durch ihren Titel und eine eventuell vorhandene Rubrik oder eine Kategorisierungsinformation dargestellt, die original aus der Dokumentensammlung entnommen wird. Für jedes Dokument listet ein *semantisches Profil* die häufigsten im Text vorkommenden Konzepte auf. Wählt man ein bestimmtes Dokument aus, so wird die Ergebnisliste durch dessen textuellen Inhalt ersetzt (Abb. 4.9). Die im

Text vorkommenden Konzepte des Netzes sind darin farblich gekennzeichnet – die Treffer im Dokument. Diese können angeklickt werden, wodurch ein Wechsel in die Konzeptumgebung des ausgewählten Begriffs vollzogen wird. Mit diesen Querverbindungen soll der Anwender dazu animiert werden, von einem Thema über das Dokument in ein weiteres Thema zu springen und somit im Netz zu stöbern.

Eine weitere Funktion, das *semantische Highlighting*, verdeutlicht den Zusammenhang zwischen den Treffern im Dokument und dem aktuellen Konzept in der Konzeptumgebung. Der mittlere Knopf unter der Überschrift *Hervorhebung* dient dazu, alle Konzepte zu markieren, die im Text das aktuelle Konzept oder einen seiner Nachfahren darstellen (s. Abb. 4.10). Dadurch kann eine Art graphischer Überprüfung stattfinden, welcher Anteil an den Treffern auf das durch das aktuelle Konzept repräsentierte Thema zurückzuführen ist. In Abb. 4.10 sind es lediglich die Konzepte *toxicology* und *organ toxicology*, die die Zuordnung des Dokuments als erweiterter Treffer von *Wissenschaftsgebiete in der Humanmedizin* begründen. Die weiteren Knöpfe heben die Markierung auf bzw. dehnen sie auf alle Treffer im Text aus, wie es in Abb. 4.9 der Fall ist.

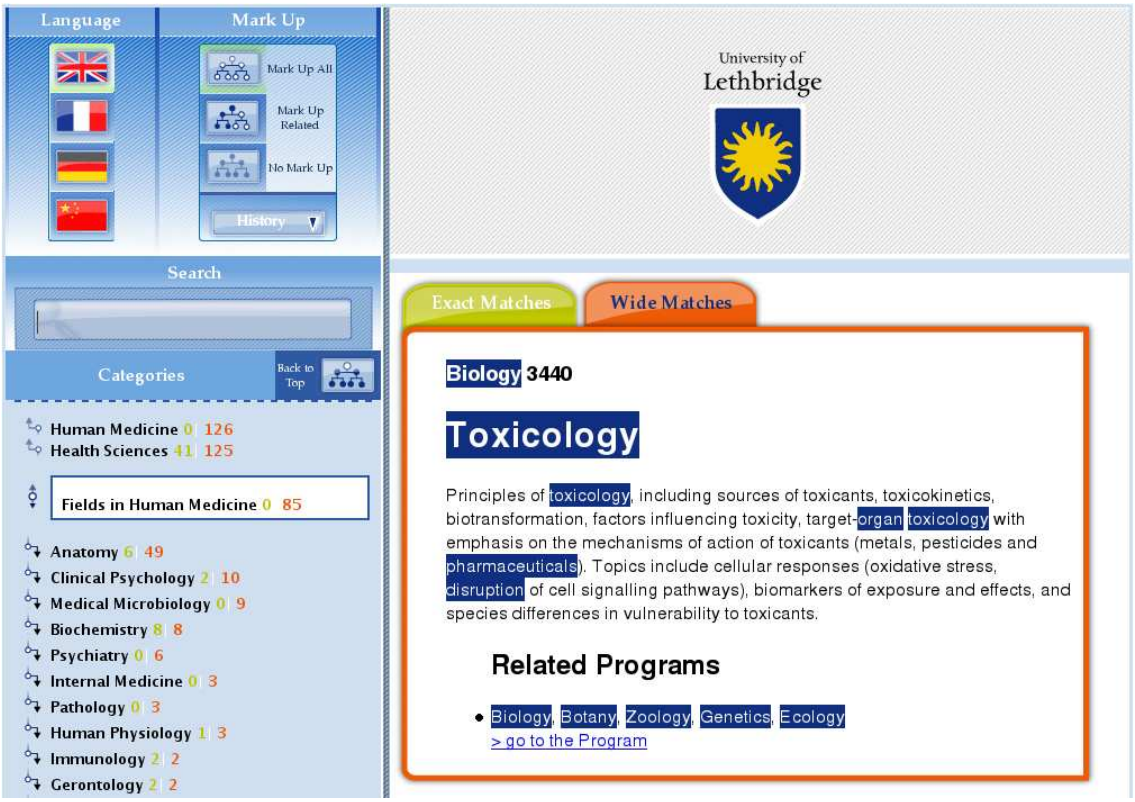


Abbildung 4.9: Dokumentdarstellung und Markierung der Treffer. Alle Treffer im Dokument erscheinen hier hervogehoben, da der erste Knopf im Bereich links oben (Mark-Up /Hervorhebung) ausgewählt wurde. Das Beispiel entstammt einer Instanz des Archivbrowsers, die auf das Vorlesungsverzeichnis der University of Lethbridge (Alberta, Kanada) aufgesetzt wurde.

Die Anzeigesprache kann mit den entsprechenden Knöpfen ausgewählt werden, wobei sich dadurch die Überschriften und die Vorzugsbezeichnung der Kategorien in der Konzeptumgebung ändern. Beispielsweise erfolgt die Anzeige in Abb. 4.10 auf Deutsch. Je nach Abdeckung der Sprache im EFGT-Netz können hier eventuell Lücken in der Darstellung der Konzeptumgebung auftauchen. In den Fällen, in denen die ausgewählte Anzeigesprache von der Dokumentsprache abweicht, können zusätzlich innerhalb des Dokuments über einen *mouse-over*-Effekt die Übersetzungen in der jeweiligen Sprache für die Konzepttreffer eingeblendet werden, so wie in Abb. 4.10 für den Treffer *pharmaceuticals*.

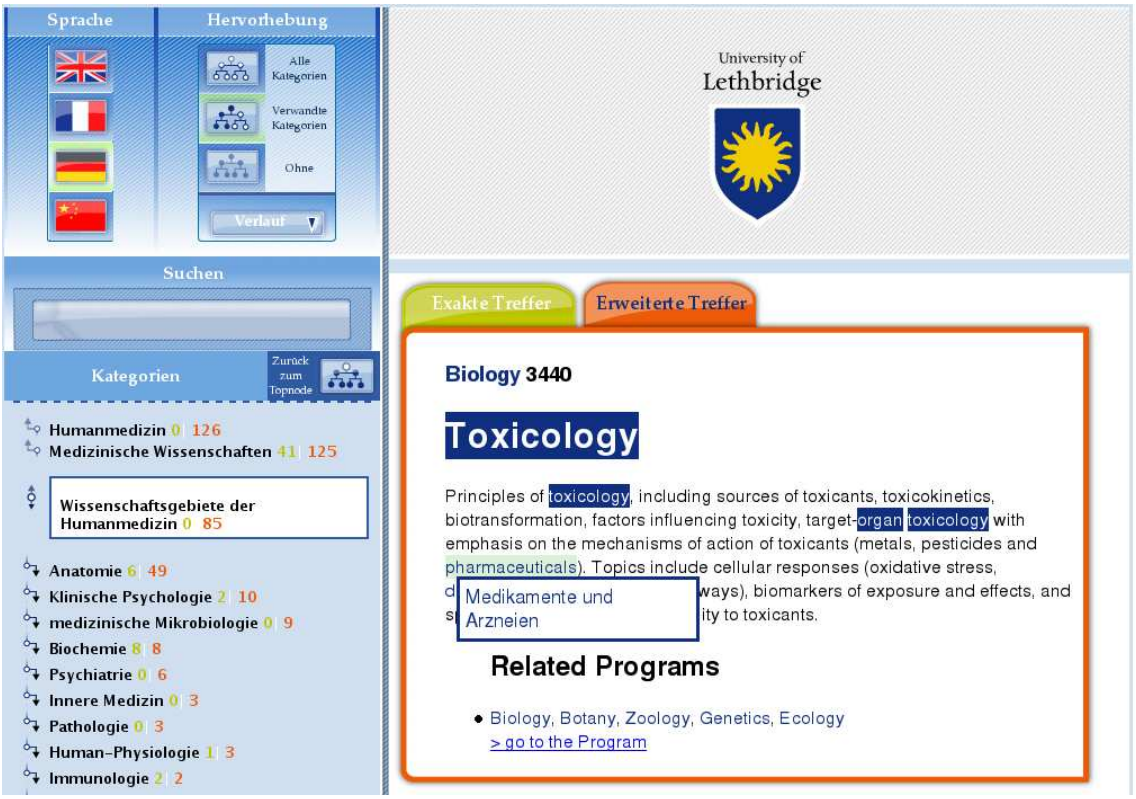


Abbildung 4.10: Semantisches Highlighting und Übersetzungsfunktionalität. Der "Verwandte Kategorien"-Hervorhebungsmodus (vgl. Abb. 4.9) wählt im Dokument die Treffer aus, die die Zuordnung des Dokuments zum aktuell angezeigten Konzept (links im Bereich Kategorien) rechtfertigen. In diesem Fall stellen die Treffer *toxicology* und *organ toxicology* die Beziehung zwischen dem Dokument und ihrem gemeinsamen Vorfahren im Netz *Wissenschaftsgebiete in der Humanmedizin* her. Das Dokument wird dadurch zum erweiterten Dokumenttreffer für dieses Konzept. Über einen Mouse-over-Effekt wurde eine Übersetzung in der aktuell gewählten Sprache (Deutsch) für den Konzepttreffer *pharmaceuticals* eingeblendet.

4.3.2 Die Architektur des thematischen Archivbrowsers

Wie bereits erwähnt, handelt es sich beim thematischen Archivbrowser um eine Client-Server-Anwendung, deren Architektur wie beim Upload-Tool ebenfalls dem Model-View-Controller-Muster folgt (s. S. 84). Eine weitere Gemeinsamkeit mit dem Upload-Tool ist die technische Infrastruktur, die ebenfalls auf dem Apache Tomcat Server und dem XNet-Interface aufsetzt. Vom EFGT-Netz-Browser-Plug-In (vgl. Weigel et al., 2006, sowie Kap. 2, S. 53) werden die später näher erläuterte Proxy-Komponente und der Dokumentparser übernommen. Die Architektur des Archivbrowsers ist in Abb.4.11 schematisch dargestellt.

Im *View*, der dem Benutzer im Client präsentiert wird, werden im Wesentlichen die Konzeptumgebung sowie dazugehörige Dokumentenlisten bzw. Dokumente dargestellt. Dementsprechend umfasst das *Modell* einerseits das Netz, das bereits als EFGT-Netz-Datenbank vorhanden ist, und andererseits die Abbildung von Konzepten auf Dokumentenmengen – das eigentliche thematische Inhaltsverzeichnis – sowie die Dokumentensammlung, die als zusätzliche Tabellen realisiert wurden. Die Tabelle, die die Dokumentensammlung repräsentiert und auf der das thematische Inhaltsverzeichnis verweist, dient zur Darstellung der Trefferlisten. Die einzelnen Dokumente sind darin mittels einer eindeutigen Identifikationsnummer und einer Reihe weiterer Merkmale wie Titel, semantisches Profil, eine URI, usw. erfasst. Details zur Tabelle, die das thematische Verzeichnis in der Datenbank hält

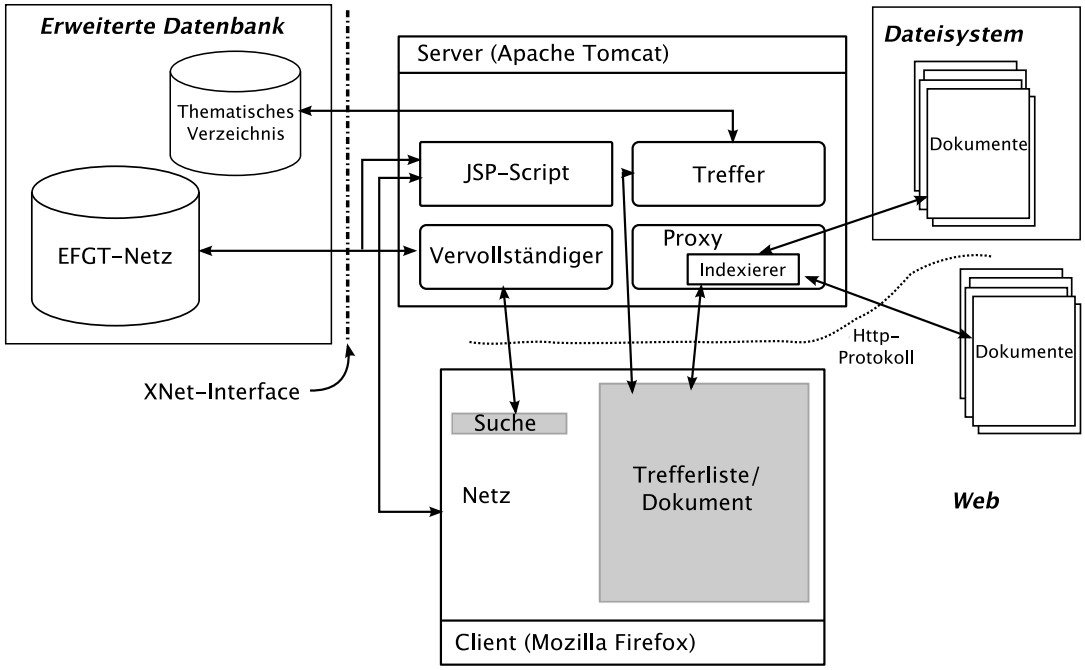


Abbildung 4.11: Architektur des thematischen Archivbrowsers. In der Clientdarstellung kommunizieren die grau unterlegten Bereiche asynchron mit dem Server; im Server sind die entsprechenden Dienste als Komponenten mit abgerundeten Ecken dargestellt.

finden sich im nachfolgenden Abschnitt 4.3.3, *Berechnung eines thematischen Inhaltsverzeichnisses*. Entsprechend dieser Erweiterung der EFGT-Netz-Datenbank wurde die Funktionalität des XNet-Interfaces (s. S. 41) ausgeweitet, um den Zugriff auf die neuen Tabellen zu ermöglichen. Die Dokumente selbst, die im HTML-Format vorliegen müssen, können entweder physisch auf der Festplatte des Servers gespeichert oder über externe URLs vom Internet angefordert werden. Bevor ein Dokument in die Client-Sicht eingebunden wird, läuft es durch eine *Proxy*-Komponente, die die im Dokument vorkommenden Konzepte des EFGT-Netzes im Text auszeichnet. Die in der Proxy-Komponente verwendeten Mechanismen zur Identifikation von Treffern bauen auf einem Dokumentparser auf, der ebenfalls vom Indexierer bei der Berechnung des thematischen Inhaltsverzeichnisses eingesetzt wird und im Abschnitt 4.3.3 näher beschrieben wird.

Der *Controller* besteht aus einem serverseitigen JSP-Skript, das das XNet-Interface anspricht und mit den so erhaltenen Daten die Sicht für den Benutzer aufbaut. Dieses Skript nimmt die durch die Aktionen des Benutzers ausgelösten Requests entgegen und baut den View entsprechend neu auf.

Im Vergleich zum Upload-Tool fällt die Architektur des thematischen Archivbrowsers wesentlich schlanker aus, da kein komplexes Modell aus Java-Objekten aufgebaut werden muss, das außerdem noch serialisiert und dann mit Hilfe eines AJAX-Mechanismus' in der Client-Sicht aktualisiert werden muss. Dementsprechend kann auf einen RPC-Mechanismus – im Fall des Upload-Tools DWR – zur Durchführung von Aktionen im Modell verzichtet werden. Während der Navigation im thematischen Verzeichnis baut der Controller bei jedem Wechsel der Konzeptumgebung den View neu auf. Ein Teil der Funktionen im Client fordern dennoch im AJAX-Stil asynchron Daten beim Server an: Der Suchmechanismus mit Auto-Vervollständigung, die Anzeige der zwei alternativen Trefferlisten zum aktuellen Konzept sowie der Dokumenten-Darstellungsmechanismus, der den Proxy anspricht. Hierfür werden im Server entsprechende Dienste zur Verfügung gestellt.

4.3.3 Berechnung eines thematischen Inhaltsverzeichnisses

Die Berechnung eines thematischen Inhaltsverzeichnisses für eine bestimmte Dokumentsammlung auf der Grundlage eines EFGT-Netzes erfolgt in verschiedenen Schritten, die in Abb. 4.12 schematisch dargestellt sind. Nach der Berechnung liegt das Verzeichnis in der erweiterten Datenbank vor und kann zur Darstellung im Client über das erweiterte XNet-Interface abgefragt werden. Ein thematisches Inhaltsverzeichnis ist vergleichbar mit einem invertierten Index, der bei einer Internet-Suchmaschine oder einem klassischen Information Retrieval (IR)-System[1] eingesetzt wird. In diesem Fall sind die Keywords oder der indexierte Wortschatz die linguistische Repräsentation der Konzepte im EFGT-Netz. Im folgenden wird auf die verschiedenen Schritte der Berechnung und die hierfür entwickelten Programme eingegangen.

Aus der Perspektive einer Methodolgie für EFGT-Netze spielt die Berechnung eines thematischen Inhaltsverzeichnisses die Rolle eines vorbereitenden Schrittes für die Adaption

[1] Ein Überblick findet sich bei Baeza-Yates und Ribeiro-Neto (vgl. Baeza-Yates and Ribeiro-Neto, 1999)

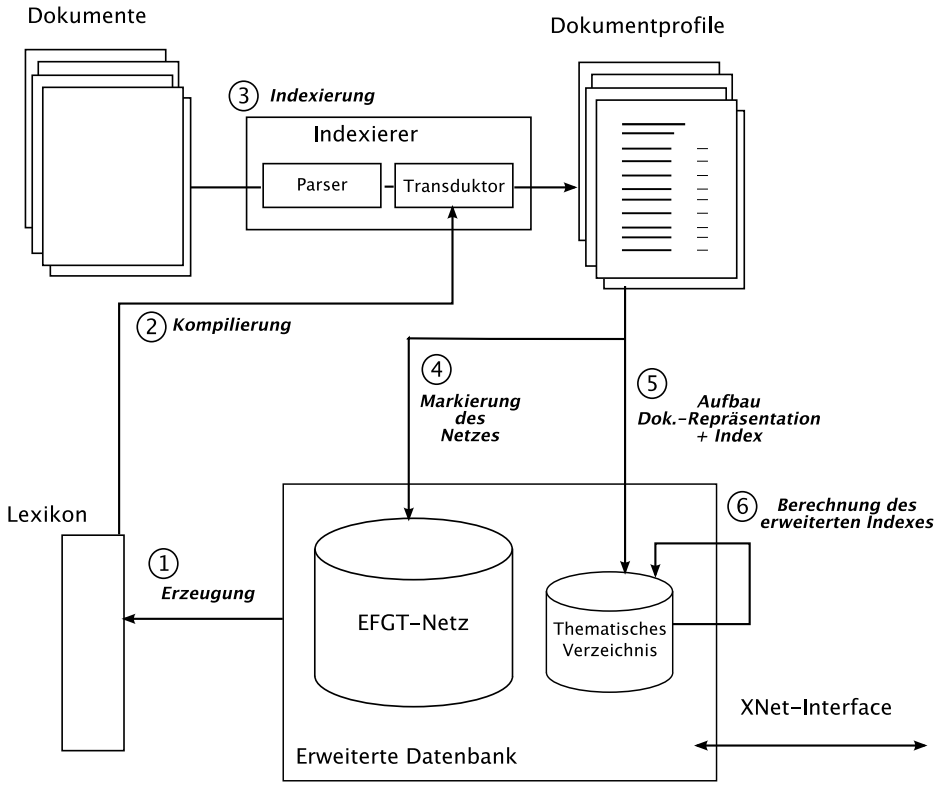


Abbildung 4.12: Phasen der Erstellung eines thematischen Inhaltsverzeichnisses

des Netzes an eine Dokumentensammlung (vgl. Kap. 2, Abschnitt 2.2.3). Mit der Indexierung wird das Teilnetz ermittelt, das in Bezug auf die betrachtete Dokumentsammlung relevant ist. Durch die physische Auslagerung dieses Teilnetzes erhält man eine spezialisierte Ressource, die sich daraufhin weiterentwickeln lässt, die durch die Dokumentensammlung dargestellte Wissensdomäne detalliert zu erfassen.

Erzeugung und Kompilierung eines Lexikons

Als Vorbereitung für die Indexierung, in der nach Konzepttreffern in den Dokumenten gesucht wird, muss ein Lexikon der linguistischen Repräsention der in der EFGT-Netz-Datenbank vorhandenen Konzepte erzeugt und kompiliert werden. Hierfür wird zunächst ein Java-Programm eingesetzt, das für die als Parameter angegebenen Sprachen – üblicherweise die Sprache, in der die Texte der Dokumentsammlung verfasst sind – eine Lexikondatei aus der Datenbank generiert (1. Schritt in Abb. 4.12). Ein Lexikondatei enthält im wesentlichen zwei Spalten. Die erste Spalte stellt einen sortierten Schlüssel dar, der über alle unterschiedlichen Ausdrücke geht, die in der Datenbank für die angegebenen Sprachen

als linguistische Repräsentation eines Konzepts angegeben sind. Für einen bestimmten Ausdruck (einen Schlüssel) wird in der zweiten Spalte eine Liste der Konzepte angegeben, die den Ausdruck in ihrer linguistischen Repräsentation enthalten. In dieser Liste sind die Konzepte durch ihre Identifikationsnummer in der Datenbank und zusätzliche Angaben wie Sprache, Konzepttyp, usw. kodiert. Wird in der Liste auf diese Weise mehr als ein Konzept kodiert, so stellt der dazugehörige Schlüssel einen *ambigen* Lexikoneintrag dar.

In einem zweiten Schritt (*Kompilierung* in Abb 4.12) wird vom selben Java-Programm ein externes aufgerufen, das einen endlichen Automaten aus den Schlüsseln der Lexikondatei aufbaut. Dieser Automat wird bei der Indexierung von einem extrem effizienten Transduktor zur Identifikation der Treffer in den Dokumenten eingesetzt (s.u.). Das Kompilierungsprogramm und der Transduktor wurden von Mihov und Schulz (vgl. Mihov and Schulz, 2007) in C++ implementiert.

Indexierung

Der wesentliche Schritt bei der Erzeugung eines thematischen Inhaltsverzeichnisses ist die Indexierung – die dritte Phase in Abb. 4.12 –, bei der die Dokumente analysiert werden und die dem Indexierungsschritt beim Aufbau eines klassischen Information Retrieval (IR) Systems oder einer Internet-Suchmaschine ähnelt. Diese Indexierung wird in diesem Fall vom sogenannten Indexierungsprogramm übernommen, das ebenfalls in Java implementiert ist. Als Ergebnis dieser Phase erhält man die sogenannten Dokumentprofile, auf deren Grundlage das thematische Inhaltsverzeichnis in der Datenbank berechnet wird. Ein Dokumentprofil listet die Konzepttreffer und deren Frequenz im spezifischen Dokument auf sowie die Daten, mit denen das Dokument in der Datenbank repräsentiert wird (Titel, Identifikationsnummer, semantisches Profil, Rubrik, usw.). Diese Daten werden in den zwei darauffolgenden Schritte (Schritte 4 und 5 in Abb. 4.12) vom Indexierungsprogramm selbst direkt in die Datenbank eingespeist.

Die Verarbeitung der Dokumente wird vom *Indexierer* durchgeführt, der sich im Wesentlichen aus zwei Komponenten zusammensetzt, dem Parser und dem Transduktor (s. Abb. 4.12). Auf diese beiden Komponenten greift ebenfalls das Proxy zurück, um in den Dokumenten die Konzepttreffer für die Darstellung im Client zu kennzeichnen. Dies garantiert, dass die Treffer, die im Client in einem Dokument dargestellt werden, konsistent mit den im Inhaltsverzeichnis Erfassten bleiben. Das ist ein wesentlicher Unterschied zur Indexierung bei einer klassischen Suchmaschine, bei der die Dokumentpositionen und weitere Merkmale der Treffer im Index erfasst werden, um später die Darstellung der Treffer im Dokument zu ermöglichen.

Der *Parser* übernimmt bei der Indexierung folgende Aufgaben:

- Normalisierung der Dokumente. Hier geht es darum, die unterschiedlichen Zeichenkodierungen, in denen die HTML-formatierten Dokumente vorliegen können, zu behandeln und insbesondere die als HTML-Entities kodierten Sonderzeichen aufzulösen.

- Extraktion der Daten zur Dokumenterfassung in der Datenbank. Angaben wie Titel, Rubrik, usw. werden aus dem Header-Teil des HMTL-Dokuments extrahiert.

Zusätzlich weist der Parser jedem Dokument eine eindeutige Identifikationsnummer zu. Diese Daten werden Bestandteil des Dokumentprofils.

- Wortidentifikation und Normalisierung. Der integrierte Tokenizer identifiziert Wortgrenzen und bringt die so ermittelten Wörter auf eine Normalform. Diese Behandlung ist von der Sprache des Dokuments abhängig, um unterschiedliche Schreibkonventionen wie etwa die Groß-/Kleinschreibung zu berücksichtigen, kann aber ingesamt als konservativ bezeichnet werden. So wurde im Englischen mit einer Stammreduktion nach dem Porter-Stemmer-Algorithmus experimentiert, dann aber verworfen, weil sie zu vielen irrelevanten Konzepttreffern geführt hatte. Die Implementierung des Tokenizers baut auf Bibliotheken des Suchmaschinen-Toolbox' Lucene (vgl. Apache Software Foundation, 2006) auf. Der Tokenizer wird auch bei der Erzeugung des Lexikons eingesetzt, um die Einträge entsprechend der Dokumentaufbereitung zu generieren.

Eine dynamische Erkennung der Sprache des Dokuments wird durch ihre explizite Angabe als Parameter des Indexierungsprogramms ersetzt.

Der *Transduktor* ist ein externes Programm, das im Indexierungsprogramm eingebunden wird. Er nimmt das vom Parser aufbereitete Dokument entgegen und ist zuständig für die Erkennung der darin vorkommenden Keywords, die mit den Konzepten im EFGT-Netz korrespondieren. Dafür setzt er den in Schritt 2 kompilierten Automaten ein. Wird mit Hilfe des Automaten im eingehenden Text ein Lexikoneintrag erkannt, ersetzt der Transduktor im eingehenden Dokument diesen durch einen bestimmten, speziell markierten Ausdruck: Dieser enthält den erkannten Lexikoneintrag sowie die mit ihm korrespondierende, in der zweiten Spalte der Lexikondatei kodierte Liste mit den entsprechenden Konzeptangaben (s. o. Lexikonerzeugung). Bei der Erkennung von Mehrwortausdrücken in den Texten wählt der Transduktor den längsten Lexikoneintrag aus, der von der aktuellen Textposition aus nach links mit den in der Eingabe vorgefundenen Wörtern übereinstimmt (*longest left match*-Strategie). Nach dem Scannen mit dem Automaten wird das modifizierte Dokument zurück an das Indexierungsprogramm geleitet, das die speziell markierten Ausdrücke filtert und sammelt. Die erkannten sprachlichen Ausdrücke und die korrespondierenden Konzept-Identifikationsnummern[2] werden vom Indexierungsprogramm im Hauptspeicher gehalten bis der komplette Text analysiert worden ist, um anschließend eine Frequenzliste und das semantische Profil für das Dokument zu berechnen. Ist das Dokumentprofil vollständig ermittelt, wird es direkt an die Datenbank übermittelt, in der die nächsten Schritte stattfinden.

[2]Die anderen Angaben, die in der Lexikondatei vorhanden sind, werden beim Einsatz des Transduktors in der Proxy-Kompenente benötigt. Das Proxy verarbeitet lediglich die durch den Transduktor speziell markierten Ausdrücke zu einer HTML-Darstellung der Konzepttreffer im Dokument weiter, in der die zusätzlichen Angaben verwendet werden.

Ermittlung des relevanten Teilnetzes

Auf der Grundlage der vom Indexierer erstellten Dokumentprofile wird der Teil des EFGT-Netzes ermittelt, der alle in der Dokumentsammlung identifizierten Konzepte abdeckt und die im thematischen Inhaltsverzeichnis aufgeführt werden sollen. Die Konzepttreffer werden vom Indexierungsprogramm über das XNet-Interface in eine spezielle Tabelle `occurrences`, s.u.) geschrieben. Bei dieser Operation wird in der Datenbank ein Trigger ausgelöst, der dafür zuständig ist, das relevante Teil des Netzes zu markieren (4. Schritt in Abb. 4.12): Für jedes Konzept, das in einem Dokument gefunden wurde, wird beim Konzept selbst und bei allen seinen Vorfahren im EFGT-Netz ein spezielles *Flag* gesetzt, die sie als relevant kennzeichnet. Bei ambigen Konzepttreffern werden alle Interpretationen des Treffers markiert. Dies bedeutet, dass alle Konzepte, für die der gefundene Ausdruck stehen kann, sowie deren jeweilige Vorfahren dann zum relevanten Teilnetz gehören.

Aufbau der Indizes und Ranking

Die Daten, die mit den Dokumentprofilen vom Indexierungsprogramm übermittelt werden, werden im 5. Schritt dafür eingesetzt, die Repräsentation der Dokumentsammlung sowie die Indextabellen aufzubauen. Diese Tabellen liegen der Darstellung der zwei Arten von Dokument-Ergebnislisten im Client zugrunde und implementieren das thematische Inhaltsverzeichnis in der Datenbank.

Die Tabelle `documents` erfasst die Daten, die die Dokumentsammlung repräsentieren. Der Schlüssel dieser Tabelle stellt die vom Indexierer zugewiesene, im Dokumentprofil angegebene Identifikationsnummer eines jeden Dokuments dar; die Attribute enthalten die aus dem Dokument extrahierten Daten, die dann im Client in der Ergebnisliste erscheinen sollen.

Die Listen der Konzepttreffer aus den Dokumentenprofilen werden in der Tabelle `occurrences` gespeichert. Diese enthält Zeilen der Form *(Dokument-ID, Konzept-ID, Frequenz)*, sodass sich für ein bestimmtes Konzept direkt die Liste seiner exakten Dokumenttreffer ablesen lässt.

Im 6. Schritt wird eine analoge Tabelle aufgebaut, `wide_occurrences`, die dazu verwendet wird, die erweiterten Treffer für ein bestimmtes Konzept zu ermitteln. Dieser Prozess wurde als Trigger in der Datenbank implementiert. In einer Zeile von `wide_occurrences` haben die angegebenen Werte eine andere Interpretation als bei `occurrences`: Im Dokument mit der in der Zeile aufgeführten *Dokument-ID* wurde ein Treffer gefunden, der im EFGT-Netz ein Nachfahre des Konzepts mit der Identifikationsnummer *Konzept-ID* ist. Die in der Zeile an der Stelle der Frequenz angegebene Zahl wird bei der Berechnung des Rankings für die Liste der erweiterten Treffer eingesetzt. Für die Definition dieser Zahl wurden verschiedene Schemata ausprobiert, die alle derselben Grundidee folgen: Für das in der Zeile aufgeführte Konzept stellt die Zahl ein Gewicht dar, das die Gewichte der im EFGT-Netz darunter liegenden Konzepttreffer im Dokument vereint. Eine Möglichkeit besteht darin, die Frequenz der Konzepttreffer im relevanten Teilnetz nach oben zu propagieren, sodass das Gewicht eines bestimmten Konzepts die Summe der Frequenzen der von

ihm abgedeckten Konzepte darstellt. Ein einfacheres Schema, das einen schnelleren Aufbau von `wide_occurrences` ermöglicht, vergibt an jedes Konzept des relevanten Teilnetzes die Anzahl der darunter liegenden Konzepttreffer.

`wide_occurrences` berücksichtigt die Struktur des Netzes und somit die Beziehungen der Konzepte zueinander, um erweiterte Treffer zu einem Konzept zu ermitteln. `wide_occurrences` ist wesentlich größer als `occurrences`, da im Prinzip jedes Dokument mit jedem Konzept des relevaten Teilnetzes in Beziehung gesetzt wird statt nur mit den Konzepttreffern im Dokument, wie es bei `occurrences` der Fall ist. `occurrences` und `wide_occurrences` übernehmen im thematischen Archivbrowser die Funktion, die in einem klassichen IR-System der invertierte Index hat.

Für die Anzeige der Dokument-Trefferlisten wurde ein sehr einfaches Ranking gewählt, bei dem das Gewichtungsschema keine wesentliche Rolle spielt und jedes Dokument j in eine fünfstufige Skala eingeordnet wird:

$$rank_i(j) = round(5 \times \frac{c_{ij}}{\max\limits_{t \in matches(i)}(c_{it})})$$

In der Formel steht i für die Konzept-ID und c_{ij} für die Zahl, die in der Zeile mit der Konzept-ID i und der Dokument-ID j in der entsprechenden Tabelle angegeben ist. t läuft über alle IDs von Dokumenten, die einen Treffer des Konzpets i darstellen. Somit bewertet dieses Ranking, wieviele Treffer jedes Dokument im Vergleich zum Dokument mit den meisten Treffern für das jeweilige Konzept hat.

Da das Hauptziel der Entwicklung des thematischen Archivbrowsers in der Unterstützung des Ontologieentwicklungsprozesses liegt, wurde kein *cut-off*-Kriterium für die Ergebnislisten definiert, um einen vollständigen Einblick in die Ergebnisse zu gewähren.

4.4 Bedeutung für den Ontologieentwicklungsprozess

Wie in der Einleitung des Kapitels erläutert, soll im Rahmen der Interaktion mit dem thematischen Archivbrowser ein qualitatives Bild über den Entwicklungsstand der Ressource gewonnen werden. Im konkreten Kontext der Anwendung sollen damit spezifische Probleme in Bezug auf die Kodierung aufgedeckt werden, die aus der abstrakten Betrachtung der Zieldomäne während des Ontologieaufbaus schwer vorauszusehen sind. Dadurch soll begleitend zum Ontologieentwicklungsprozess insgesamt eine qualitative Qualitätskontrolle stattfinden. Die vom thematischen Archivbrowser bereitgestellte Funktionalität kommt dieser Zielsetzung durchaus nach, auch wenn sie nicht spezifisch dafür entworfen wurde, eine Evaluation zu ermöglichen, sondern vielmehr in Hinblick auf den Einsatz in einer Endanwendung ausgearbeitet wurde:

Qualitative Bewertung der Netzstruktur

- Über die Navigation im relevanten Teil des EFGT-Netzes kann überprüft werden, ob die Struktur der Ontologie eine akzeptable Modellierung der Konzeptbeziehungen in

der Wissensdomäne erfasst. Im Vergleich zur Navigation in der Entwicklungsoberfläche (vgl. Kap. 2, S. 45) stellt dies eine natürliche, zielgerichtete Aktivität dar – man will ja dadurch zu relevanten Dokumenten gelangen – , bei der die Netzstruktur unter einer bestimmten Perspektive betrachtet wird. Außerdem erfolgt diese Aktivität ausschließlich auf den relevanten Teil der Struktur, wodurch spezifische Details des EFGT-Netzes im Vordergrund stehen. Bewegt man sich im thematischen Inhaltsverzeichnis von oben nach unten mit dem Ziel, ein bestimmtes Konzept zu finden, so muss bei jedem Schritt eine Auswahl aus vorhandenen Konzepten getroffen werden. Dadurch wird die mittels der Netzstruktur ausgearbeitete Modellierung der Domäne überprüfbar: Sind die an jedem Schritt gemachten Unterscheidungen intutitiv und gelangt man dadurch zum anvisierten Konzept, so ist sie im Einklang mit der eigenen Konzeptualisierung. Ähnlich vermittelt die Anzahl der Schritte, die für das Erreichen eines bestimmten Konzepts nötig ist, oder das Ausprobieren verschiedener zur Verfügung stehenden Wege ein qualitatives Bild der Struktur.

- Für ein spezifisches Konzept kann mit Hilfe des semantischen Highlightings überprüft werden, ob die jeweils darunter gefassten Konzepte als thematisch zusammengehörig empfunden werden. Dieser Mechanismus lässt sich außerdem auf unterschiedlichen Ebenen der Ontologie einsetzen, d.h. für eine Reihe von allgemeineren bis hin zu spezifischeren Konzepten, sodass in einem bestimmten, fixierten Dokument jeweils unterschiedliche "Projektionen" des Netzes beobachtet werden können. Betrachtet man die in einer solchen Projektion enthaltenen Konzepte zusammen mit dem, von dem die Projektion ausgegangen ist, so kann man einerseits beurteilen, ob die Konzepttreffer darin als thematisch zusammengehörig empfunden werden, und andererseits, ob die Assoziationsketten von den Konzepttreffern zum projizierten Konzept nachvollziehbar wirken. Beides ermöglicht eine qualitative Bewertung der Netzstruktur.

Einschätzung der Abdeckung. Ein Bild über den Grad der Erschließung des Wissensbereichs, der durch die indexierte Dokumentsammlung repräsentiert wird, kann mit Hilfe der im thematischen Browser bestehenden zweifachen Relation zwischen Dokumenten und Konzepten gewonnen werden:

- Über die exakten Dokumenttreffer lassen sich Texte besichtigen, in denen ein bestimmtes Konzept vorkommt. Dadurch kann ein optischer Eindruck der Abdeckung des Bereichs gewonnnen werden, in dem das spezifische Konzept eingebettet ist. Je nachdem, ob in den Texten viele Konzepttreffer beobachtet werden können bzw. offensichtlich wird, dass viele für den Bereich wichtige benannte Entitäten nicht erfasst sind, kann man von einer besseren oder schlechteren Abdeckung ausgehen. Exakte Dokumenttreffer bieten gleichzeitig die Möglichkeit, Beziehungen zwischen einem Konzept und mit ihm in exakten Treffern korrelierenden zu beobachten, sodass weitere Details der Domäne sichtbar werden.

- Die Beziehung eines Konzepts zu seinen erweiterten Treffern lässt sich als – wenn auch rudimentäre – Kategorisierungsfunktion auffassen, in der die mit dem Konzept

verbundene Netzstruktur einbezogen wird. Sind etwa Texte vorhanden, die als besonders repräsentativ für ein bestimmtes Konzept gelten, so lässt sich testen, ob diese unter dem jeweiligen Konzept als erweiterte Treffer aufgeführt werden und ob sie an prominenter Stelle des Rankings erscheinen. Sind die repräsentativen Texte richtig kategorisiert – d.h. haben sie einen hohen Rang bzgl. des ausgewählten Konzeptes –, spricht dies für eine angemessene Abdeckung sowie für eine sinnvolle Modellierung des durch das Konzept repräsentierte Thema. Erscheint wiederum ein Dokument unter einem bestimmten Konzept als völlig falsch kategorisiert, so ist das meistens auf eine geringe Abdeckung des eigentlichen Themas des Dokuments oder auf Effekte wie unaufgelöste Ambiguitäten zurückzuführen.

In der Praxis hat sich herausgestellt, dass die Interaktion mit einem EFGT-Netz im Rahmen des thematischen Archivbrowsers entscheidend dazu beiträgt, frühzeitig auf Probleme bei dessen Entwicklung aufmerksam zu machen. Beispielsweise wurde begleitend zur Entwicklung des CoGE-Netzes eine Sammlung von Zeitungsartikeln indexiert und mit Hilfe des Archivbrowsers darin navigiert. So wurde daraufhin entdeckt, dass es eine Überlappung zwischen den Namen vieler deutscher Gemeinden, die in einem Schritt zuvor mit Hilfe des Upload-Tools integriert wurden, und dem allgemeinen deutschen Wortschatz gibt. Insgesamt betrachtet konnte durch den Einsatz des thematischen Browsers der Ausbau des CoGE-Netzes fokussierter ablaufen. Auch war die Navigation in der Sammlung von Zeitungsartikeln in vielen Fällen für die Entscheidung ausschlaggebend, einen bestimmten thematischen Bereich auszubauen. Ebenfalls durch die Navigation im thematischen Inhaltsverzeichnis wurde man auf wichtige oder sehr aktuelle Themen aufmerksam.

Wie bereits angesprochen (s. S. 122) stellt die Indexierung einer Dokumentensammlung einen vorbereitenden Schritt dar, wenn es darum geht, aus einer allgemeinen Ontologie wie dem CoGE-Netz einen relevanten Teil auszukoppeln, aus dem dann in einer Phase der Adaption (s. Kap 2, S. 54 ff) eine spezialisierte, auf die Dokumentsammlung abgestimmte Ontologie entwickelt wird.

Erfahrungen mit einer solchen Adaptionsphase wurden in der Praxis im Rahmen eines Projektes gemacht, das als Ziel hatte, mit Hilfe des Archivbrowsers einen semantischen Zugang auf das Vorlesungsverzeichnis einer kanadischen Universität bereitzustellen (vgl. Torres Schumann et al., 2008). Die Abbildungen 4.9 und 4.10 zeigen die dazugehörige Variante des thematischen Browsers. Als Vorbereitung für die Adaption wurde zunächst das Vorlesungsverzeichnis auf der Grundlage des CoGE-Netzes indexiert und das relevante Teilnetz physisch ausgelagert. Am Anfang der Adaptionsphase zeigte sich, dass durch die anschließende Navigation im Archivbrowser eine Reihe von Problemen entdeckt werden konnten, die sich aus der Einschränkung auf die spezifische Domäne ergaben. So stellte sich vor dem Hintergrund eines Vorlesungsverzeichnisses die obere Teilhierarchie des CoGE-Netzes als zu allgemein bzw. zu abstrakt heraus. Es zeigte sich ebenfalls, dass die Modellierung bestimmter gängiger Konzepte wie *Gold* oder *Mensch*, die gerade im CoGE-Netz aus einer allgemeinen Perspektive kodiert wurden, im Fall eines Vorlesungsverzeichnisses an das Verständnis einzelner Fächer wie Bergbau oder Philosophie anzupassen war; ähnlich war dies der Fall für ambige Konzepte. Während der tatsächlichen Adaptionsphase lieferte der

Archivbrowser ein wichtiges Feedback zur Beurteilung über den Fortschritt der Arbeiten. In diesem Projekt war dies besonders der Fall, weil die eigentliche Motivation für die Adaption darin bestand, die Navigation im Archivbrowser an die anvisierte Dokumentensammlung zu optimieren.

Zusammenfassend lässt sich sagen, dass die Bereitstellung einer Anwendung, innerhalb derer ein spezifisches EFGT-Netz eingesetzt wird, zur Steuerung der Entwicklungs- bzw. Adaptionsphase beiträgt, da dadurch frühzeitig Anregungen geliefert bzw. spezifische Probleme aufdeckt werden. Im Rahmen einer Methodologie, in der ansonsten keine systematische Evaluation durchgeführt wird, ergibt sich daraus eine wichtige Funktion für eine praktische Anwendung.

4.5 Der thematische Browser als Navigationsmittel für Pressearchive

Die hier vorgestellte thematische Suche und Navigation mit Hilfe des Archivbrowsers stellt eine neuartige Ergänzung der Navigationsmechanismen dar, die sich aktuell in der Webpräsenz von Presseanbietern beobachten lassen, und hat Merkmale, die besonders für die Archivverwertung relevant sind.

Die Struktur des thematischen Inhaltsverzeichnisses stellt eine Gliederung des Archivs dar, die von sehr allgemeinen Kategorien bis hin zu detaillierten Ansichten einzelner Wissensbereiche reicht. Insbesondere wird dadurch eine *hierarchische Navigation* ermöglicht, die tiefer greift als die eher flachen Strukturen der üblichen Rubriken-Navigation. Dies kann eine sinnvolle Ergänzung zu den übrigen aktuell eingesetzten Mechanismen (Querverweise, verwandte Kategorien, usw.) sein, die hauptsächlich eine assoziative, netzartige Navigation im Archiv erlauben.

Die hierarchische Struktur des thematischen Inhaltsverzeichnisses kann für den Benutzer eine wichtige Unterstützung sein, wenn er sich in einem thematischen Bereich nicht auskennt; sie liefert ihm wichtige Stichwörter für die Suche. Zudem sind wegen des Aufbaus des thematischen Inhaltsverzeichnisses alle darin vorkommenden Konzepte für das Archiv relevant. Mit einem thematischen Inhaltsverzeichnis wird eine sinnvolle Alternative zur Volltextsuche bereitgestellt, die nur dann eine besonders effiziente Zugangsmöglichkeit darstellt, wenn a priori klar ist, welche Informationen gesucht werden. Ein weiterer Vorteil der Struktur des EFGT-Netzes kann darin gesehen werden, dass Begriffe über verschiedene Wege erreicht werden können und somit die darunter klassifizierten Dokumente aus verschiedenen thematischen Perspektiven zugänglich gemacht werden.

Die strukturelle Navigation im thematischen Archivbrowser wird mit einer assoziativen über die im Dokument hervorgehobenen Konzepttreffer kombiniert; sie dienen als Querverweise und führen zur zentralen Struktur zurück. Damit entsteht ein geschlossener Kreis, der zum Stöbern und Navigieren im Archiv ermuntert und der Schaltung von Werbeinhalten einen erweiterten Raum bietet. Die Kennzeichnung von Konzepttreffern im Dokument hat außerdem die Funktion, die Einordnung des Dokuments unter ein bestimmtes Thema

zu belegen und transparent für den Benutzer zu machen. Dies wird durch das semantische Highlighting verstärkt, das den Anteil eines Themas am Dokument sichtbar macht. Mechanismen zur thematischen Kategorisierung, die intuitiv nachvollziehbar sind, sind für die Akzeptanz des Systems durch die Benutzer entscheidend.

Durch die Kombination verschiedener Navigationsmodelle – strukturell, assoziativ, mit Hilfe von Filtern/ Suche – wird im Archivbrowser eine reichhaltige, ausgewogene Navigationserfahrung (Kalbach, 2008, S. 10 ff) umgesetzt, die unterschiedlichen Benutzerbedürfnissen entgegenkommt: Überblicksgewinnung, Unterstützung in unbekannten Wissensbereichen, Suche nach spezifischen Informationen. Damit wird ein System von Zugangsmöglichkeiten eröffnet, das auf eine bessere Sichtbarkeit von Archivinhalten abzielt.

Unabhängig von der spezifischen Umsetzung als thematischer Browser kann das CoGE-Netz als grundlegende Ressource angesehen werden, die sich über die Realisierung einer umfangreichen Archivnavigation hinaus für die automatische Beschlagwortung und das semantische Tagging von Dokumenten einsetzen lässt. Damit kann ein Teil der manuellen Arbeit entfallen, die es bei der Archivaufbereitung zu leisten gilt, und es kann eine neue Qualität der Erschließung erreicht werden, auf denen Funktionalitäten der Dossiererstellung und Textklassifikation aufbauen können.

Bei der Weiterentwicklung des thematischen Archivbrowsers zu einem kommerziellen Produkt sind verschiedene zusätzliche Funktionen und Verbesserungen denkbar. Dazu gehört etwa eine Erweiterung der Suchfunktion durch eine komplementäre Volltextsuche oder durch die Möglichkeit, mehrere Konzepte zu kombinieren und damit etwa eine konjunktive Anfrage zu stellen. Darüber hinaus ließe sich das semantische Profil zur Charakterisierung eines Dokuments in Abhängigkeit des aktuell im Verzeichnis aufgeschlagenen Konzepts berechnen, um das im Dokument unter dem Gesichtspunkt des Konzepts Spezifische für den Benutzer sichtbar zu machen. Eine Evaluation unterschiedlicher Rankingschemata sowie die Definition eines Schwellenwertes für die Zuordung von Dokumenten zu einzelnen Konzepten werden als wichtige Maßnahmen auf dem Weg zu einem kommerziellen Produkt erachtet.

Unter dem Aspekt einer kommerziellen Anwendung stellt die hier vorgestellte Implementierung des Dokumentenindex, die eine Datenbanktabelle mit einer Proxy-Komponente zur Dokumentendarstellung kombiniert, einen sehr flexiblen Indexierungsmechanismus dar. So lassen sich ohne größeren Aufwand einzelne Dokumente aus dem Index herausnehmen oder hinzufügen, sodass sich ein thematisches Inhaltsverzeichnis nicht nur für eine statische, sondern auch für eine sich verändernde Dokumentensammlung aktuell halten lässt, z.B. eine Sammlung neuester Nachrichten.

4.6 Schlussfolgerungen und Ausblick

Mit dem thematischen Archivbrowser wurde eine konkrete Anwendung entwickelt, in der sich das Zusammenspiel eines EFGT-Netzes mit einer bestimmten Dokumentensammlung beobachten lässt. Die vom Archivbrowser bereitgestellten Funktionen ermöglichen, ein detailliertes Bild des Entwicklungszustands der Ontologie in Bezug auf den indexierten Kor-

pus zu gewinnen. In dieser Hinsicht spielt die hier vorgestellte Anwendung eine wichtige Rolle als Ersatz für eine systematische Evaluation, mit der ein zur Steuerung des Entwicklungsprozesses entscheidendes Feedback erhalten wird. Darüber hinaus lässt sich die Aufbereitung einer Dokumentensammlung als ein erster Schritt in der Erstellung einer spezialisierten Ressource auffassen.

Als Endanwendung erstellt der Archivbrowser automatisch ein thematisches Inhaltsverzeichnis und ermöglicht dadurch die Navigation in einem Dokumentenarchiv. In diesem Zusammenhang hat der Archivbrowser innovative Merkmale, die unter dem Gesichtspunkt eines kommerziellen Einsatzes, etwa zur Verwertung von Pressearchiven, relevant sind. Die darin realisierte Navigation kombiniert assoziative und hierarchische Elemente, die zum Stöbern in der Dokumentensammlung animieren, zugleich aber eine Orientierung im Archiv geben. Welchen Effekt dies genau auf das Verhalten der Nutzer hat und wie sich dies in Kennzahlen ausdrückt, die für die Schaltung von Online-Werbung ausschlaggebend sind, lässt sich nur in einer empirischen Studie herausfinden. Problematische Aspekte des Archivbrowsers sind sowohl seine manchmal als zu groß empfundene Tiefe des Navigationsraums als auch einzelne Konzepte, die zwar als Sammelbehälter vieler Konzepte wie *Aids in Afrika* dienen, jedoch wie *Gesundheitsthemen nach Kontinenten* wenig intuitiv sind.

Der thematische Archivbrowser lässt sich in verschiedene Richtungen weiterentwickeln, etwa um eine Funktion zu ermöglichen, mit der Benutzer Feedback über die Interaktion mit der Ontologie an die Entwickler weiterleiten können, oder durch ein verbessertes Dokumentranking. Auf die spezifischen Möglichkeiten, die der Archivbrowser durch den Bezug auf einen Korpus für die Akquisition neuer Konzepte und für die Maintenance der Ontologie bietet, wird im nächsten Kapitel näher eingegangen.

Kapitel 5

Dokumentzentrierte Akquisition von Konzepten: *Maintenance* im Browser-Editor

5.1 Motivation

Die Verfügbarkeit des im letzten Kapitel entwickelten Archivbrowsers bedeutet für den Lebenszyklus eines EFGT-Netzes, dass ein wesentlicher Schritt der Qualitätskontrolle stattfinden kann. Durch die Interaktion mit dem Archivbrowser kann implizit der Entwicklungszustand eines EFGT-Netzes in Bezug auf die Wissensdomäne – die für eine spezifische Dokumentensammlung steht – überprüft werden. Dadurch können Aspekte des Verhältnisses zwischen Ontologie und Wissensdomäne aufgedeckt werden, die relevant für die Entwicklung der Ontologie sind und auch deren weiteren Ausbau maßgeblich beeinflussen können. Vorranging sind das etwa Probleme oder Fehler bei der Modellierung von Konzepten oder mangelnde Abdeckung der Wissensdomäne.

Während der Navigation in der Dokumentensammlung mit Hilfe des thematischen Archivbrowsers können aber auch Ausdrücke und Phrasen in den Texten beobachtet werden, die einen Verweis auf noch nicht erfasste Konzepte darstellen. Hierbei erscheinen diese Kandidaten für neue Konzepte jeweils in einem bestimmten *semantischen Kontext*, der durch die Konzepte gebildet wird, die auf der Oberfläche des thematischen Archivbrowsers gleichzeitig mit dem Kandidaten angezeigt werden. So wird im Text ein bestimmter Kandidat in der Regel von anderen erkannten Konzepten des EFGT-Netzes umgeben; gleichzeitig wird auf dem Archivbrowser eine bestimmte Region des thematischen Inhaltsverzeichnisses dargestellt, über die der betreffende Text erreicht wurde. Dieser Kontext, in den der Kandidat eingebettet ist, liefert einen Hinweis auf dessen thematische Zugehörigkeit und ermöglicht somit, ihn zumindest grob semantisch einzuordnen.

Aus dieser Überlegung heraus ist es naheliegend, den thematischen Archivbrowser für eine dokumentenzentrierte Akquisition neuer Konzepte einzusetzen. Hierbei stellt sich zunächst die Frage, wie sich die Funktionalität des Archivbrowsers so erweitern lässt, dass

beobachtete Kandidaten sich leicht im Rahmen der Interaktion mit dem Archivbrowser in das betreffende EFGT-Netz integrieren lassen. Dies wirft weitere Fragen auf:

- Wodurch lassen sich die Relationen gewinnen, die als Grundlage für die semantische Modellierung der neuen Konzepte nötig sind?
- Kann zumindest ein Teil dieser Relationen dem Kontext entnommen werden, in dem der Kandidat im Archivbrowser erscheint?
- Können nach demselben Prinzip auch sprachliche Varianten erfasst werden, mit denen sich die linguistische Repräsentation bestehender Konzepte im Netz erweitern lässt?
- Wie lässt sich der Prozeß der Akquisition neuer Konzepte in der Dokumentensammlung systematisieren, etwa mit Hilfe der automatischen Identifikation von Kandidaten?

Unabhängig von der Antwort auf diese Fragen hat ein erweiterter Archivbrowser, der sich zur Akquisition neuer Konzepte in einer Dokumentensammlung einsetzen lässt, eine strategische Bedeutung in einer Methodologie zum Aufbau von EFGT-Netzen. Sowohl bei der Entwicklung als auch bei der regelmäßigen Pflege oder bei der Adaption eines bestehenden Netzes an eine geschlossene Domäne bedeutet die Akquisition neuer Konzepte eine zentrale vorbereitende Aktivität. Darüber hinaus sind Textdokumente eine besonders wichtige Datenquelle, wenn es um die Erfassung benannter Entitäten und eines themenspezifischen Vokabulars geht. Ein erweiterter Archivbrowser, in dem sich beobachtete Kandidaten leicht in das Netz integrieren lassen, wäre in diesem Zusammenhang eine Möglichkeit, in der eine explorative, dokumentzentrierte Akquisition stattfinden kann. Ohne diese Möglichkeit ist man auf die Kodierungsoberfläche angewiesen, in der der Kandidat aus dem Kontext herausgerissen wird, oder es müssen semistrukturierte Daten bereitgestellt werden, mit denen sich die Entwicklung eines EFGT-Netzes mit Hilfe des Upload-Tools vorantreiben lässt – in den meisten Fällen würde dies bedeuten, dass eine separate Phase der Wissensakquisition durchgeführt werden muss. Eine solche Möglichkeit dokumentenzentrierter Akquisition ist besonders relevant, wenn es darum geht, die Aktualität des Netzes fortlaufend zu sichern: Es ist davon auszugehen, dass neue Ereignisse oder aufkommende benannte Entitäten zunächst in einem Text erwähnt werden, bevor sie in (semi-)strukturierter Form systematisch etwa in einer Datenbank erfasst werden. Somit stellt im Lebenszyklus eines EFGT-Netzes ein um Akquisitionsfunktionalität erweiterter Archivbrowser eine wichtige Ergänzung zum Upload-Tool dar, die einen fundamentalen Beitrag zur fortlaufenden Pflege oder zur systematischen Erfassung einer geschlossenen Domäne leisten kann.

In diesem Kapitel wird eine Konzeption für die Erweiterung der Funktionalität des thematischen Archivbrowsers erarbeitet mit dem Ziel, die Akquisition und Integration von Kandidaten im Rahmen der Navigation zu ermöglichen. Im folgenden verweisen die Namen Browser-Editor oder auch dokumentenzentrierter Editor auf diese Erweiterung des Archivbrowsers.

Als Grundanforderung für den entworfenen Browser-Editor soll die Konzeptakquisition durch die Interaktion mit seiner Benutzeroberfläche möglich sein. Diese soll die Funktionalität des thematischen Archivsbrowsers umfassen und erlauben, Konzeptkandidaten zu markieren, in Zusammenhang mit bestehenden Konzepten zu bringen und somit in das Netz zu integrieren. Der Effekt, auf diese Weise das Netz zu erweitern, soll sich sofort im Archivbrowser – etwa im Ranking – beobachten lassen. Damit bekommt der Ontologieeditor ein unmittelbares, qualitatives Feedback über seine Arbeit. Dabei soll weniger von einer Verwendung durch einen naiven Benutzer ausgegangen werden als vielmehr durch einen Ontologieentwickler, der den EFGT-Formalismus kennt und etwa in der Lage ist, das Upload-Tool zu bedienen.

Das Kapitel ist in zwei Teile untergliedert. Im ersten Teil (Abschnitt 5.2) wird zunächst begründet, warum die Akquisition neuer Konzepte aus einer methodologischen Betrachtung heraus ein sinnvoller erster Schritt für den Ausbau des thematischen Archivbrowsers sein kann sowie die Voraussetzungen betrachtet, die in einem entsprechenden Browser-Editor für die Durchführung dieser Aufgabe gegeben sind. Im zweiten Teil des Kapitels (Abschnitt 5.3) wird eine genauere Konzeption für den Browser-Editor herausgearbeitet. Eine Beschreibung der nötigen Schritte für die Implementierung dieser Konzeption dient dabei als eine Art Machbarkeitsstudie, mit der die Umsetzbarkeit des konzipierten Browser-Editors belegt wird.

Zur verwendeten Terminologie: Ein *Konzeptkandidat* ist im folgenden ein sprachlicher Ausdruck, der in einem Text der Dokumentsammlung beobachtet und als Verweis auf ein im EFGT-Netz noch nicht erfasstes (semantisches) Konzept wahrgenommen wird. *Semantischer Kontext eines Konzeptkandidaten* bezeichnet die Konzepte des EFGT-Netzes, die auf der Oberfläche des thematischen Archivbrowsers zu dem Zeitpunkt dargestellt werden, zu dem der Konzeptkandidat beim Navigieren beoabachtet wird.

5.2 Anforderungen an die Funktionalität des Browser-Editors

5.2.1 Verschiedene Möglichkeiten für den Ausbau des thematischen Archivbrowsers

Befasst sich man aus einer allgemeinen Perspektive heraus mit der Idee, eine ontologiebasierte Anwendung so auszubauen, dass man darin die Entwicklung und Plege der Ontologie durchführen kann, so ergeben sich eine Reihe unterschiedlicher Möglichkeiten für den konkreten Ausbau. Welche Funktionen hierbei relevant sind, hängt stark mit der Rolle zusammen, die je nach verfolgter Methodologie die Anwendung im Lebenszyklus der Ontologie einnimmt. Im Laufe dieser Arbeit wurde an verschiedenen Stellen auf diese Rolle verwiesen:

- Bei den meisten der in Kap. 1 erwähnten Methodologien stellt die Anwendung einen konkreten Rahmen bereit, in dem sich die Ontologie tatsächlich einsetzen und testen

lässt. Im Lebenszyklus erfolgt somit ein Schritt der Qualitätskontrolle – ebenfalls eine wesentliche Motivation für die Entwicklung des thematischen Archivbrowsers in Kap. 4. In den meisten Methodologien schließt sich an den Evaluationsschritt eine Phase der Weiterentwicklung oder der Maintenance an, die in der Regel durch spezialisierte Ontologieentwickler schon außerhalb der Anwendung stattfindet. Hier stellt sich die Frage, ob zumindest ein Teil dieser Aufgaben sinnvoll innerhalb der Anwendung unterstützt werden kann.

- Andere der in Kap. 1 erwähnten Methodologien (s. S. 22 ff.) sehen jedoch in der Anwendung eine Plattform, in der die Konsensbildung und die Entwicklung der Ontologie durch Endnutzer des jeweiligen Anwendungsbereichs stattfinden kann. Dementsprechend machen diese Methodologien Vorschläge für die Erweiterung der Anwendungsfunktionalität, um einen wesentlicher Teil der Entwicklung darin abzudecken.

- Speziell beim thematischen Archivbrowser wird ein Bild über den Zustand der Ontologie in Bezug auf eine bestimmte Dokumentensammlung vermittelt, sodass naheliegt, die Funktionalität des Archivbrowsers so zu erweitern, bei Bedarf die Adaption des relevanten Teilnetzes an die Dokumentsammlung durchführen zu können.

Im Prinzip ergeben sich somit unterschiedliche Möglichkeiten für den Ausbau des thematischen Archivbrowsers, je nachdem, ob man kollaborative Szenarien berücksichtigen, den Endnutzer an der Entwicklung der Ontologie beteiligen will oder die fortlaufende Pflege bzw. die Adaption der Ontologie in einem bestimmten Bereich im Vordergrund stehen soll.

Wie lassen sich die wesentlichen Anforderungen identifizieren, die einen möglichst flexiblen Einsatz des erweiterten Archivbrowsers innerhalb des Lebenszyklus' eines EFGT-Netzes erlauben? Dieser Frage wird in den folgenden Abschnitten nachgegangen.

5.2.2 Fokus auf Akquisition

Tabelle 5.1 fasst spezifische Aufgaben zusammen, die im Fall eines bestehenden EFGT-Netzes bei der Durchführung einer *Maintenance*- bzw. Adaptionsphase auf der Grundlage einer Dokumentensammlung anfallen. In dieser Gegenüberstellung ist erkennbar, dass die Akquisition neuer Konzepte die einzige Aufgabe ist, die sich im Wesentlichen in beiden Szenarien auf die gleiche Weise durchführen lässt. Dagegen haben sowohl die Behandlung von Ambiguitäten als auch die Anpassung der Netzstruktur unterschiedliche Ziele in beiden Szenarien. Das spricht dafür, den Archivbrowser zunächst nur um die Funktionalität zu erweitern, die die Akquisition ermöglicht. Dadurch wird zum einen vermieden, den erweiterten Archivbrowser auf den Einsatz in einer der beiden Phasen zu spezialisieren. Zum anderen wird der für die Akquisition vom Archivbrowser spezifisch gebotene Vorteil ausgenutzt, der darin besteht, Konzeptkandidaten in einem bestimmten semantischen Kontext zu beobachten und somit zunächst thematisch einordnen zu können. Im Gegensatz dazu hat etwa die Darstellung der Netzstruktur im Archivbrowser keine spezifischen Merkmale,

die eine Funktionalitätserweiterung in Hinsicht auf die Editierung der Netzstruktur rechtfertigen würde.

Ein weiterer Grund, die Erweiterung des thematischen Archivbrowsers zunächst auf die Akquisition zu fokussieren, ist die in der Einleitung erwähnte strategische Bedeutung, die Wissensakquisition auf der Grundlage von Textdokumenten für den Lebenszyklus eines EFGT-Netzes hat.

Ein neues Konzept zu akquirieren bedeutet einerseits, dessen linguistische Repräsentation zu gewinnen, und anderseits, die für die semantische Modellierung grundlegenden Beziehungen zu anderen Konzepten zu kennen. Setzt man sich zum Ziel, im Rahmen des thematischen Archivbrowsers die Akquisition neuer Konzepte zu ermöglichen, so muss man näher untersuchen, wie sich damit diese beiden Arten von Informationen gewinnen lassen.

Fortlaufende Pflege, Maintenance	**Adaption an geschlossene Domäne**
Akquisition neuer aufkommender Konzepte (ergänzende *Population*)	Akquisition relevanter spezifischer Konzepte (spezifische *Population*)
Verfeinerung und Aktualisierung der Modellierung bestehender Konzepte	Anpassung der Modellierung bestehender Einträge an Betrachtungsweise der Domäne
Vornehmen kleinerer Korrekturen bzw. Ergänzungen der Netzstruktur	Anpassung der Netzstruktur je nach Ziel: • Zur Erfassung der Domäne im angemessenen Detaillierungsgrad, etwa durch Elimination von zu allgemeinen oder zu spezifischen Ebenen • Für eine spezielle Anwendung: etwa, um eine möglichst effiziente Navigation im Netz zu ermöglichen.
Behandlung von Ambiguitäten: Sammeln von Informationen, bspw. Kontextbedingungen, die als Grundlage für Disambiguierungsverfahren dienen.	Auflösung von Ambiguitäten: Identifikation der in der eingeschränkten Wissensdomäne spezifischen Lesarten

Tabelle 5.1: Gegenüberstellung von Aufgaben bei der *Maintenance* bzw. bei der Adaption von EFGT-Netzen

5.2.3 Akquisition der linguistischen Repräsentation

Im Archivbrowser können benannte Entitäten und sprachliche Ausdrücke, die auf neue Konzepte verweisen, dadurch entdeckt werden, mit Hilfe des thematischen Inhaltsverzeichnisses in der Sammlung zu navigieren und Dokumente zu sichten. Insbesondere lässt sich die Funktionalität des Archivbrowsers so einsetzen, dass auf Dokumente hingewiesen wird, für die das verwendete EFGT-Netz eine geringe Abdeckung bietet und in denen wahrscheinlich ist, noch nicht erfasste Konzepte zu finden. Hierfür besteht beispielsweise die

Möglichkeit, für ein spezifisches Konzept die Liste von erweiterten Treffern zu prüfen und nach aufffällig gerankten Dokumenten zu suchen, wie in Abschnitt *Einschätzung der Abdeckung* (Kap. 4, S. 128 ff) diskutiert. Für die Handhabung der auf diese Weise gefundenen Kandidaten in der Benutzeroberfläche des Browser-Editors bedarf es dann eines entsprechenden Mechanismus', um sie zu markieren und manipulierbar zu machen.

Die Akquisition von Konzeptkandidaten könnte weitaus effizienter stattfinden, indem Methoden eingesetzt werden, die entsprechende sprachliche Ausdrücke auf der Grundlage ihrer computerlinguistischen Eigenschaften automatisiert identifizieren und diese für den Benutzer optisch kenntlich machen, indem sie im Text speziell ausgezeichnet sind. Über eine zusätzliche, spezielle Zugangsmöglichkeit im Archivbrowser könnte man dann eine systematische Inspektion der Kontexte ermöglichen, in denen die erkannten Kandidaten auftreten.

Für die Akquisition relevanter sprachlicher Ausdrücke kann man auf eine Reihe von Verfahren zurückgreifen, die in der Computerlinguistik und angewandten Feldern wie IR oder Informationsextraktion angewendet werden und u.a. auf die Erkennung benannter Entitäten (*named entity recognition*) oder des spezifischen Vokabulars einer Domäne (Erkennung von Terminologie) in Textdokumenten abzielen. Dazu zählen Methoden, die sich auf statistische Merkmale dieser Ausdrücke verlassen, wie etwa einfache Frequenzlisten, N-Gramme, Messungen von Entropie, usw. sowie Verfahren, die auf linguistischen Eigenschaften aufbauen – angefangen bei einfachen regulären Ausdrücken über lokale Grammatiken bis hin zur Erkennung bestimmter Folgen grammatischer Kategorien, usw.

Zu einer Konzeption des Browser-Editors gehört es, eine Auswahl dieser Verfahren zu treffen, die sich in die vorhandene Architektur integrieren lassen. In diesem Zusammenhang wird hier die These aufgestellt, dass der Browser-Editor die erwähnten Verfahren in dem Sinne ergänzt, dass er eine semantische Einordnung der damit erhaltenen Ergebnisse ermöglicht.

Unabhängig von der automatischen Identifikation von Kandidaten ist es sinnvoll, im Browser-Editor als grundlegenes Akquisitionsverfahren die Möglichkeit zu behalten, Kandidaten händisch zu markieren und zu bearbeiten, um eventuelle Unzulänglichkeiten des eingesetzten automatischen Erkennungsverfahrens zu korrigieren.

5.2.4 Akquisition von Relationen für die Kodierung

Neben der Akquisition der linguistischen Form ist für die Integration eines neuen Konzeptes in ein bestehendes EFGT-Netz nötig, das neue zu modellieren und in Relation zu bestehenden Konzepten zu setzen, d.h. mittels der Angabe passender Komponenten im ID-String das Konzept zu kodieren. Unter welchen Voraussetzungen lässt sich diese Aufgabe im angedachten Browser-Editor erfüllen?

Ein wesentlicher Faktor ist die Dokumentsammlung, die im Browser-Editor bereitgestellt werden soll. Mit dem Aufbau eines EFGT-Netzes wird ein Teil des Hintergrundwissens erfasst, von dem in einer bestimmten Domäne ausgegangen wird (s. Kap. 2, S. 35). Es ist anzunehmen, dass in allen Texten, die einer Domäne zuzuschreiben sind, nur in Einzelfällen auf dieses Hintergrundwissen explizit hingewiesen wird, etwa in Texten, in

denen die Domäne selbst oder einzelne Begriffe dieser zum Thema gemacht werden. In der Domäne des Allgemeinwissens könnten dies bspw. Artikel einer Enzyklopädie sein. So ist davon auszugehen, dass im Browser-Editor die Akquisition von Komponenten für die Modellierung erleichtert wird, wenn die entsprechenden semantischen Relationen explizit im Text angesprochen werden. In diesem Fall könnte man diese einfach ablesen und durch die Bereitstellung geeigneter Mechanismen im Browser-Editor während der Navigation direkt übernehmen. Somit ist eine wichtige Voraussetzung hierfür, dass die bereitgestellte Dokumentensammlung einen entsprechenden, repräsentativen[1] Korpus darstellt.

Im anderen Fall muss man darauf setzen, dass der vom Browser-Editor bereitgestellte semantische Kontext, in dem die Konzeptkandidaten beobachtet werden, die für die Modellierung nötigen Relationen enthält. Wie wahrscheinlich das ist, lässt sich ohne einschlägige Tests kaum voraussagen. Im schlimmsten Fall lässt sich der Kandidat nur grob thematisch einordnen, indem ihm ein geeignetes Konzept aus dem semantischen Kontext zugewiesen wird. Dies könnte etwa ein Konzept sein, bei dem der fragliche Konzeptkandidat in einem seiner erweiterten Dokumenttreffer beobachtet wurde und somit den entsprechenden Text thematisch kategorisiert. Im günstigeren Fall können dagegen die relevanten Relationen am Kontext abgelesen – z.B. an den im Text den Kandidaten umgebenden Konzepten oder an der dargestellten Netzstruktur – und für die Modellierung übernommen werden. In beiden Fällen soll im Browser-Editor dem Benutzer möglich sein, diese Informationen zu speichern: Im ungünstigen Fall kann die grobe thematische Einordnung der Kandidaten als Grundlage für mächtigere, in einer separaten Phase eingesetzte Methoden zur Wissensakquisition genutzt werden; im günstigen Fall lassen sich Hilfe der gesammelten Daten die entsprechenden Einträge im EFGT-Netz durchführen.

Darüber hinaus lässt sich für die Akquisition der nur implizit gegebenen Relationen darauf setzen, dass sich diese durch bestimmte Eigenschaften manifestieren, etwa durch ein gehäuftes gemeinsames Auftreten des Kandidaten und der betreffenden Konzepte in den Texten. Entsprechend kann man die Möglichkeit in Erwägung ziehen, im Browser-Editor Mechanismen einzubinden, die Hinweise auf implizite Relationen liefern. Die Extraktion semantischer Relationen aus Textkorpora stellt ein lebendiges Forschungsfeld dar, sodass hierfür ein breites Spektrum von Verfahren vorhanden ist. Ein Ansatz, der sich im Browser-Editor leicht realisieren und integrieren lässt, besteht darin, Korrelationen zwischen dem Kandidaten und bestehenden Konzepten zu ermitteln und dem Benutzer anzuzeigen. Korrelationen liefern einen Hinweis auf das Bestehen einer semantischen Beziehung und können auf der Grundlage des gemeinsamen Auftretens in Texten berechnet werden. Über die Anzeige von Korrelationen im Browser-Editor lässt sich ein alternativer Zugang zu den einzelnen Texten einbauen, um den Kontext des gemeinsamen Auftretens zu sichten.

Die Zuordnung von linguistischen Varianten zu bestehenden Konzepten baut auf der Erkennung einer bestimmten semantischen Beziehung auf, der Synonymie, und kann als

[1] In der Computerlingustik wird die Frage, was ein repräsentativer Korpus ist und wie sich ein solcher zusammenstellen lässt, oft diskutiert. Die Antwort hängt meistens mit der Aufgabenstellung zusammen; manchmal bleibt die Frage auch offen. In dem hier besprochenen Fall würde "repräsentativ" bedeuten, dass die gesuchten Relationen, die die Kodierung der Konzepte der betrachteten Domäne erlauben, im Korpus textuell explizit angesprochen werden.

spezieller Fall betrachtet werden, für den sich die hier beschriebenen Mittel ebenfalls einsetzen lassen. So kann man darauf setzen, in einem bestimmten Text neben dem bekannten Bezeichner zusätzliche sprachliche Varianten für dasselbe Konzept zu finden oder diese über Korrelationen zu entdecken.

Die Fragestellung, wann die beobachteten Relationen allgemein in der Domäne gelten oder nur punktuell in der betrachteten Dokumentensammlung zutreffen, kann alleine mit den vom Browser-Editor bereitgestellten Mitteln nicht entschieden werden, selbst wenn eine statistische Information wie Korrelationen einen Beitrag zur Klärung liefern kann. Diese Frage wurde in dieser Arbeit bereits in der Diskussion angesprochen, wann eine Änderung in einem EFGT-Netz *Maintenance* und wann *Adaption* darstellt (s. S. 54 ff) – sie hängt eng mit der Problematik der Zusammenstellung eines repräsentativen Korpus' zusammen. Die endgültige Klärung dieser Frage wird hier ausgeklammert, da dies den Rahmen an dieser Stelle sprengen würde.

5.3 Eine Konzeption für die Funktionalität des Browser-Editors

Nachdem die Hauptanforderung auf die Akquisition neuer Konzepte eingegrenzt wurde und dementsprechend auf unterschiedliche Möglichkeiten der Akquisition der linguistischen Repräsentation und der semantischen Relationen im Browser-Editor näher eingegangen wurde, wird in diesem Abschnitt eine Spezifikation für eine konkrete Implementierung erarbeitet. Diese Spezifikation umfasst zum einen eine Konzeption der Benutzeroberfläche, die die Funktionalität des thematischen Archivbrowsers integrieren soll und mit deren Hilfe beobachtete Konzeptkandidaten sich leicht während der Navigation im Archiv in das EFGT-Netz integrieren lassen sollen. Zum anderen wird ein Mechanismus entworfen, mit dem existente Verfahren zur Erkennung relevanter sprachlicher Ausdrücke während der Indexierung eingebunden werden können, um anschließend die erkannten Ausdrücke im Browser-Editor als Konzeptkandidaten dem Benutzer zur Verfügung zu stellen.

Da die Funktionalität des thematischen Archivbrowsers im Browser-Editor mit eingeschlossen werden soll, wird die Architektur des ersteren als Grundlage für eine Implementierung des letzteren betrachtet. Die Beschreibung der nötigen Erweiterungen dieser Architektur dient im letzten Teil dieses Abschnittes als Analyse, mit der die Machbarkeit des Browser-Editors gezeigt werden soll.

5.3.1 Die Benutzeroberfläche

Abb. 5.1 zeigt eine schematische Darstellung der Benutzeroberfläche des Browser-Editors. Die obere Hälfte mit den Bereichen *Netzstruktur* und *Ergebnisliste/Dokumentanzeige* stellt die Funktionalität des thematischen Archivbrowsers zur Verfügung. Die Bereiche in der unteren Hälfte bieten spezifische Funktionen des Browser-Editors an. Der linke untere Bereich dient dazu, die Ergebnisse der Akquisition zu verarbeiten, d.h. darin lassen sich beobachtete Kandidaten mit Konzepten, die auf der Oberfläche gesammelt wurden, in Beziehung

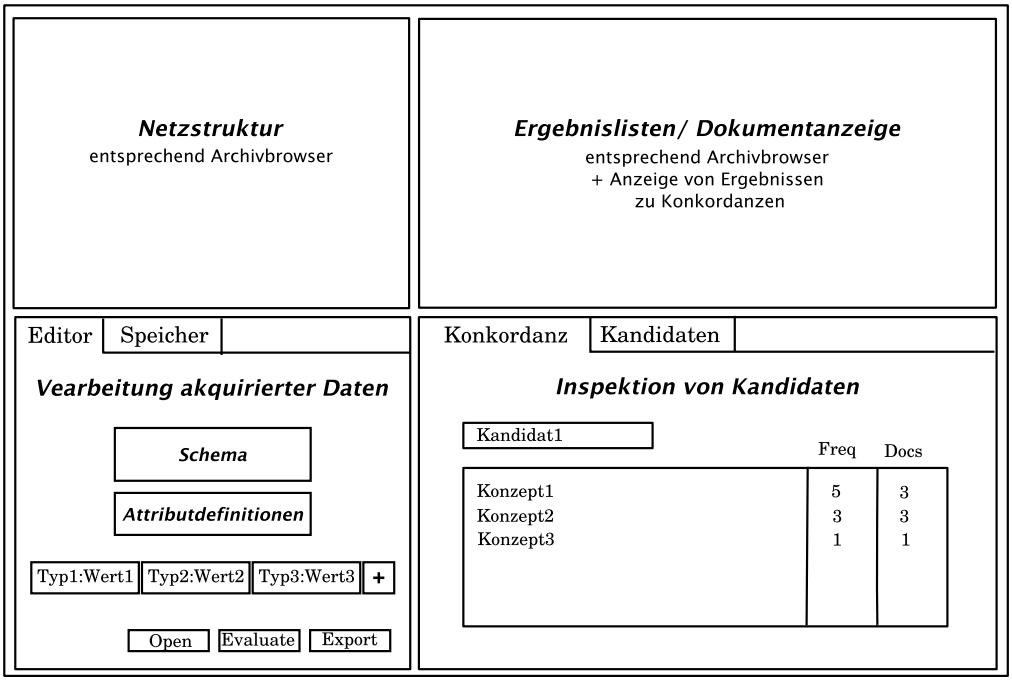


Abbildung 5.1: Schematische Darstellung der Benutzeroberfläche des Browser-Editors. Kursive Schrift deutet die Funktionalität des jeweiligen Bereichs an; normale Schrift wird zur Darstellung von Elementen und Daten auf der Oberfläche verwendet.

setzen und entweder abspeichern oder direkt dem EFGT-Netz hinzufügen. Der Bereich zur *Inspektion von Kandidaten* (unten rechts) listet die aktuell bekannten Kandidaten auf und zeigt Korrelationen zwischen Konzepten und Kandidaten an. Diese Funktion wird hier Konkordanz genannt, da sie ebenfalls den Zugang zu den textuellen Stellen ermöglicht, in denen ein Kandidat mit einem spezifischen Konzept gemeinsam vorkommt.

Um den Zugriff des Benutzers auf den aktuell angezeigten semantischen Kontext zu ermöglichen, ist ein grundlegendes Funktionsprinzip der Benutzeroberfläche, dass sich alle angezeigten Konzepte und Kandidaten graphisch, d.h. mit der Maus markieren und per Drag-and-Drop auf die unteren Bereiche ziehen lassen.

Im folgenden wird der Funktionsumfang der verschiedenen Bereiche auf Abb. 5.1 näher spezifiert.

Anzeige der Netzstruktur

Dieser Bereich entspricht weitgehend der Funktion zur Navigation im Netz des Archivbrowsers. Der Unterschied besteht nämlich darin, dass man die als Teil der Netzstruktur

angezeigten Konzepte, die ja einen Teil des semantischen Kontextes ausmachen, mit der Maus markieren und in andere Bereiche der Oberfläche ziehen kann. Die Knöpfe für das semantische Highlighting und für die Sprachauswahl werden ebenfalls in diesem Bereich angezeigt.

Anzeige von Ergebnislisten bzw. Dokumenten; Handhabung von Kandidaten

Ergebnislisten für erweiterte und exakte Dokumententreffer eines im Netz ausgewählten Konzeptes sollen wie im Archivbrowser im rechten (oberen) Bereich der Benutzeroberfläche dargestellt werden. Ein zusätzlicher Reiter in diesem Bereich soll dazu dienen, zu den einzelnen Paaren in der Konkordanz die entsprechenden Dokumentlisten anzuzeigen.

Die Konzepte, die in den Dokumentlisten als semantisches Profil der einzelnen Dokumente erscheinen, machen einen Teil des semantischen Kontexts aus und sollen per Drag-and-Drop handhabbar sein.

Als wesentlicher Akquisitionsmechanismus soll in der Darstellung der Texte die Handhabung der Konzeptkandidaten ermöglicht werden. Dabei sollen sprachliche Ausdrücke, die bereits während der Indexierung automatisch erkannt wurden, im Text grapisch kenntlich gemacht werden und gemäß dem grundlegenden Funktionsprinzip der Oberfläche mit der Maus manipulierbar sein. Zusätzlich soll die Möglichkeit bestehen, im Text interaktiv sowohl Kandidaten als auch *Stoppwörter* händisch zu markieren. Dies könnte etwa dadurch erfolgen, dass man eine Textpassage mit der Maus auswählt, der anschließend mit Hilfe eines über die rechte Maustaste aktivierbaren Kontextmenus der jeweilige Typ (Kandidat oder Stoppwort) zugewiesen wird. Daraufhin soll der markierte Bereich farblich hervorgehoben werden.

Händisch markierte sollen gleichwertig mit automatisch erkannten Kandidaten sein, d.h. genauso mit der Maus handzuhaben, und außerdem noch in der ganzen Dokumentsammlung gültig sowie über die Konkordanzfunktion und die Kandidatenliste erreichbar sein.

Die interaktive Auszeichnung von Stoppwörtern auf dem Text dient dazu, Wörter kenntlich zu machen, die keine Kandidaten darstellen und somit bei der Akquisition nicht relevant sind. Dies sind etwa Wörter, die nicht spezifisch für die betreffende Wissensdomäne sind, Funktionswörter, usw.[2] Indem man einen Bereich des Textes mit der Maus auswählt und als Stoppwörter deklariert, sollen die darin enthaltenen Wörter in der gesamten Dokumentkollektion als Stoppwörter erkannt und bei der Anzeige eines Dokuments farbig unterlegt werden. Das Ziel ist, Textpassagen graphisch einzugrenzen, die Kandidaten enthalten können und somit die interaktive Suche nach ihnen effizienter zu gestalten (s. Abb. 5.2).

[2]Die Bezeichnung Stoppwörter hier ist aus dem Bereich des Information Retrieval entnommen, in dem Wörter benannt werden, die sehr häufig auftreten und keine Relevanz für die Erfassung des Dokumentinhalts besitzen, sodass sie in der Regel bei einer Volltextindexierung nicht beachtet werden.

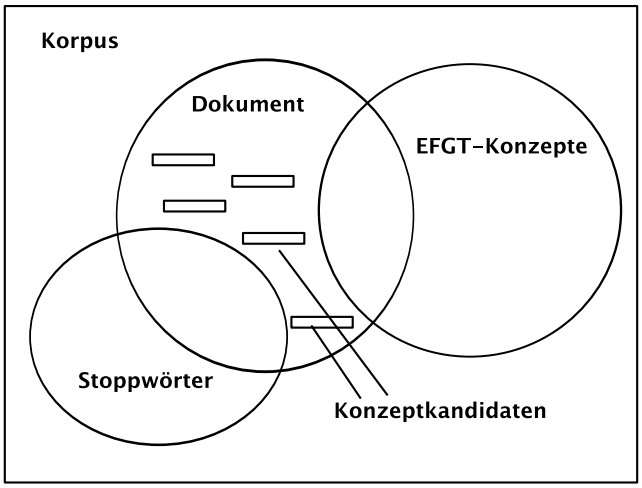


Abbildung 5.2: Verhältnis zwischen EFGT-Konzepten, Konzeptkandidaten, Stoppwörtern und sonstigen Ausdrücken in einem im Browser-Editor verfügbaren Korpus

Inspektion von Kandidaten und Konkordanzfunktion

Die in diesem Bereich bereitgestellten Funktionen dienen über die Trefferlisten hinaus als alternative Zugangsmöglichkeit auf die Dokumentensammlung, sodass sich gezielt auf die Kontexte zugreifen lässt, in denen ein spezifischer Kandidat erscheint.

Der in der semantischen Darstellung in Abb. 5.1 skizzierte Reiter *Kandidaten* soll dazu dienen, die Liste aller – d.h. sowohl händisch markierten als auch automatisch erkannten – Kandidaten zur Verfügung zu stellen und deren Häufigkeit im Korpus anzuzeigen. Indem man einen Kandidaten aus dieser Liste auswählt, soll man zu den Dokumenten gelangen können, in denen er gefunden wurde.

Die über den Reiter *Konkordanz* erreichbare Funktion soll für einen bestimmten Kandidaten Konzepte und zusätzliche Kandidaten ermitteln, die im Korpus häufig mit ihm in einem Dokument vorkommen. Die Interaktion mit dem Benutzer soll folgendermaßen stattfinden: Um eine Konkordanz anzuzeigen, wird der Kandidat auf das vorhandene Feld (s. Abb. 5.1) gezogen, woraufhin eine Liste der Konzepte und zusätzlichen Kandidaten erscheint, die gemeinsam mit dem fraglichen vorkommen. Sie sind nach der Häufigkeit des Konkordanzpaares in der Sammlung geordnet; die Anzahl der unterschiedlichen Dokumente wird ebenfalls angegeben. Klickt man auf einen Eintrag der Liste, so wird im rechten oberen Bereich der Zugang zu den Dokumenten ermöglicht, in denen das betreffende Paar zu beobachten ist.

Die Konkordanzfunktion zielt darauf, für die einzelnen Kandidaten einen Hinweis auf Konzepte zu liefern, zu denen eine Beziehung bestehen könnte. Dadurch, dass die entsprechenden Kontexte sich im Browser-Editor überprüfen lassen, soll die Relevanz des Paares

für die Kodierung des Kandidaten bewertet werden können und gleichzeitig die Konzeptakquisition gezielt stattfinden.

Konzepteditor und Relationenspeicher

Der linke untere Bereich ist dafür gedacht, Kandidaten und bestehende Konzepte, die während der Inspektion von Kontexten und der Navigation im Archivbrowser akquiriert werden, explizit miteinander in Beziehung zu setzen und weiter zu verarbeiten. In diesem Bereich sollen hierfür ein *Konzepteditor*, mit dem sich neue EFGT-Netz-Einträge durchführen lassen sowie ein sogenannter *Relationenspeicher*, in dem sich akquirierte Kandidaten und Relationen sammeln lassen, zur Verfügung gestellt werden.

Für die Spezifikation von Relationen wird auf das Format "typisierte Felder" für Upload-Files aus Kap. 3 (s. S. 77) zurückgegriffen. Die Editierung einzelner Zeilen in diesem Format soll mit einem graphischen Mechanismus möglich sein, mit dem sich die mit der Maus gezogenen Konzepte und Kandidaten auf entsprechende Felder platzieren lassen sollen.

Beim Konzepteditor – in Abb. 5.1 als Reiter *Editor* dargestellt – dient dieser graphische Mechanismus der Spezifikation der Relationen, die als Grundlage für die Modellierung eines Kandidaten eingesetzt werden. Die entsprechende Zeile, die sich als eine Reihe typisierter Felder auffüllen lässt, ist in Abb. 5.1 schematisch dargestellt: Der mit "+" gekennzeichnete Knopf fügt ihr ein neues Feld hinzu, woraufhin zunächst ein Typ für das Feld anzugeben ist. Die Spezifikation eines Werts hierfür soll durch verschiedene Eingabeoptionen möglich sein:

- Markierte Kandidaten oder Konzepte werden mit der Maus in das Feld hineingezogen;
- Ein vorhandenes Konzept wird direkt aus dem Netz ausgewählt; der angedachte Mechanismus hierfür ähnelt dem der Suchfunktion mit Autovervollständigung im Archivbrowser (s. S. 118);
- Der Wert wird händisch angegeben.

Durch ein Kontextmenu, das mit der rechten Maustaste aufgeklappt wird, kann etwa der gewünschte Eingabemechanismus ausgewählt werden.

Für die Übersetzung der Eingabezeile in einen Eintrag des EFGT-Netzes wird auf die Templatesprache des Upload-Tools zurückgegriffen. Wie im Upload-Tool (s. Kap. 3) sollen in entsprechenden Feldern ein Schema und eine dazugehörige Attributdeklaration editierbar sein. Analog zum Upload-Tool sollen Schemata sowohl direkt auf der Oberfläche editierbar sein als auch sich hochladen und exportieren lassen. Auf die Auswertung des Schemas hin soll eine Darstellung des Ergebnisses wie im Upload-Tool erscheinen und der neue Eintrag in das EFGT-Netz ladbar sein. Nach der Integration eines bestimmten Kandidaten soll dieser in der Benutzeroberfläche als bestehendes Konzept sowie dessen Auswirkungen auf die Funktionalität des thematischen Inhaltverzeichnisses (Anpassung des Rankings, semantisches Highlighting, usw.) im Browser-Editor erkennbar sein.

Durch den Einsatz der Templatesprache werden im Browser-Editor beide anvisierten Fälle, nämlich sowohl der der Akquisition von Bezeichnern zur Erweiterung der linguistischen Repräsentation bestehender Konzepte als auch der der Akquisition von Konzepten zur Kodierung eines beobachteten Kandidaten einheitlich behandelt.

Der in Abb. 5.1 als Reiter *Speicher* dargestellte Relationenspeicher soll intern einem Upload-File im Format "typisierte Felder" entsprechen, das man während des Stöberns im Archiv mit Daten auffüllt. Die einzelnen Zeilen der Datei sollen dabei mit demselben graphischen Mechanismus wie die Input-Zeile im Editor aufgebaut werden. Mittels eines Knopfes soll sich der Inhalt des Speichers als Upload-File exportieren und separat abspeichern lassen. Das exportierte File kann dann im Upload-Tool eingesetzt werden oder zur weiteren Vervollständigung in die Wissensakquisitionsphase geschickt werden.

Mögliche Erweiterungen und offene Fragen

Auf der Grundlage der hier vorgestellten Spezifikation für die Benutzeroberfläche sind verschiedene Weiterentwicklungen des Browser-Editors denkbar. So lässt sich im Prinzip der allgemein einsetzbare Konzepteditor durch eine Reihe spezieller Formulare ersetzen, die intern mit unterschiedlichen Templates korrespondieren und verschiedene Fälle abdecken, wie der Erweiterung der linguistischen Repräsentation eines bestehenden Konzepts oder der Kodierung eines gefundenen Kandidaten. Denkbar ist es auch, auf ähnliche Weise ein spezielles Formular zu entwickeln, mit dem naive Benutzer ohne Kenntnis der ID-String-Sprache Einträge im EFGT-Netz durchführen könnten. Der Browser-Editor ließe sich dann als ontologiebasierte Anwendung betrachten, in der sich Endnutzer an der Entwicklung der Ontologie aktiv beteiligen können.

In der bisherigen Spezifikation wird der Zugang auf die textuellen Kandidatenvorkommen sowohl über das thematische Inhalts- als auch über das Kandidatenverzeichnis und die Konkordanzfunktion ermöglicht. Dementsprechend ergeben sich im Prinzip verschiedene Kombinationsmöglichkeiten für die mit beiden Zugangsarten verbundene Funktionalität: Soll das Highlighting des Konkordanzpaares mit dem semantischen Highlighting kombinierbar sein? Macht es Sinn, Konkordanzen für den aktuellen Knoten in der Konzeptumgebung anzuzeigen, bzw. ein Zusammenspiel zwischen der Konkordanzfunktion und dem Navigieren im Netz zu implementieren?, usw.. Die Frage der Nützlichkeit dieser Kombinationen wird hier offen gelassen, da sie sich nur mit entsprechenden Tests in der Praxis angemessen beantworten lässt.

5.3.2 Erkennung von Kandidaten während der Indexierung

Wie in Abschnitt 5.2.3 erläutert, existieren eine Reihe von computerlinguistischen Verfahren zur Erkennung sprachlicher Ausdrücke wie benannte Entitäten oder terminologische Ausdrücke, die sich als Verweise auf Konzepte einer Ontologie verstehen lassen. Eine Eingrenzung der Methoden, die während der Aufbereitung der Dokumentensammlung für den Browser-Editor hierfür in Frage kommen, erfolgt anhand praktischer und technischer

Gründe. Ein wesentlicher Faktor hierbei ist die Tatsache, dass das Aufbereitungsverfahren auf dem Indexierungsprogramm des Archivbrowsers (s. Kap 4, S. 122 ff) aufbauen soll.

Das Kernstück des Indexierungsprogramms ist der Indexierer (s. S. 124), der sich bei der Erkennung der linguistischen Repräsentation der Konzepte im EFGT-Netz auf den sogenannten Transduktor verlässt (s. S. 125). Aufgrund der Implementierung des Indexierungsprogramms sind externe Erkennungsprogramme einzubinden, die wie der Transduktor die erkannten sprachlichen Ausdrücke direkt in dem betreffenden Dokument markiert. Damit kann auch das Proxy weiterhin auf dem ausgezeichneten Dokument aufbauen, um die Dokumentdarstellung für den Webbrowser zu generieren (s. S. 121). Die eingesetzten Verfahren sollen darüber hinaus eine ähnliche Performanz wie die des Transduktors gewährleisten. In der Praxis bedeutet dies eine Einschränkung auf Verfahren, die wie der eingesetzte Transduktor auf *finite state*-Methoden[3] aufbauen.

Aus den benannten Gründen kommen für eine erste Implementierung des Browser-Editors Verfahren zur Erkennung von Konzeptkandidaten in Betracht, die endliche Zustandsautomaten einsetzen und die erkannte Kandidaten im Text auszeichnen. Folgende Zusammenstellung zeigt eine mögliche Auswahl:

- Transduktoren zur Erkennung von Kandidaten aus einer vorhandenen Liste. Ist eine Liste von Kandidaten vorhanden, nach denen spezifisch in der Dokumentsammlung gesucht werden soll, lässt sich mit dem vorhandenen Kompilierungsprogramm (s. S. 124) ein entsprechender Transduktor aufbauen. Die vorhandene Liste spielt hierbei die Rolle des Lexikons.

- Automatenbasierte Teilstringsuche. Für die Erkennung bestimmter Kandidatenarten ist eine Teilstringsuche manchmal nützlich, z.B. im Deutschen als rudimentäre Vorgehensweise zur Identifikation von Komposita, die den gesuchten Teilstring als Stamm haben.

- Reguläre Ausdrücke lassen sich zu einem Automaten kompilieren und anschließend im Rahmen eines Transduktors einsetzen, der die entsprechenden sprachlichen Muster als Kandidat im Dokument auszeichnet.

- Lokale Grammatiken (LGs; vgl. Paumier, 2003; Gross, 1997) sind bei der lexikonbasierten Erkennung benannter Entitäten besonders leistungsfähig. LGs lassen sich mit Hilfe des Programms Unitex (vgl. Paumier, 2004) definieren und zu einem Transduktor kompilieren. Sie lassen eine hohe syntaktische Variabilität der gesuchten Ausdrücke zu, wodurch sie der vielfältigen Erscheinungsformen von Personen-, Firmennamen, Rollenbezeichnern, usw. in Texten gerecht werden und dadurch klar im Vorteil gegenüber einer normalen Stringsuche sind.

Eine Konfigurationsdatei soll dem Benutzer ermöglichen, verschiedene der oben genannten, externen Erkennungsverfahren auszuwählen und im Indexierungsprogramm einzubinden.

[3]Ein Überblick über *finite state*-Methoden oder auch endliche Zustandsautomaten bietet Mohri (vgl. Mohri, 1997). Eine umfassende Literaturliste findet sich bei XEROX (vgl. XEROX, 2009).

5.3.3 Eine Architektur für den Browser-Editor und Umsetzung der Implementierung

Bei einer Implementierung des Browser-Editors ist es sinnvoll, von der Architektur des thematischen Archivbrowsers auszugehen, da dessen Funktionalität in die neue Anwendung integriert werden soll. In diesem Abschnitt werden die Erweiterungen und Änderungen beschrieben, die an dieser Architektur nötig sind, um den spezifizierten Funktionsumfang im Browser-Editor anzubieten. Mit dieser Beschreibung wird die Realisierbarkeit der bisher entwickelten Spezifikation aus einer technischen Perspektive untersucht.

Mit der Architektur des thematischen Archivbrowsers als Grundlage (s. Kap. 4, S. 121) ergibt sich für den Browser-Editor ebenfalls eine *Model-View-Controller*-Architektur (vgl. S. 84). An dieser Stelle wird ein kurzer Überblick über die notwendigen Änderungen gegeben, auf die die nachfolgenden Abschnitte näher eingehen.

Das Modell, das im Archivbrowser in der EFGT-Netz-Datenbank verwaltet wird und im wesentlichen das Netz, die Dokumentsammlung und das thematische Inhaltsverzeichnis umfasst, muss entsprechend der Funktionalität des Browser-Editors erweitert werden, um zusätzlich Kandidaten, Stoppwörter sowie Konkordanzdaten verwalten zu können (s.u., Abschnitt *Erweiterung der Datenbank*). Darüber hinaus ist das ursprünglich beim Archivbrowser eingesetzte Indexierungsprogramm anzupassen, damit ausgehend von der gegebenen Dokumentensammlung der Aufbau des erweiterten Modells und die Erkennung der Kandidaten mit Hilfe der vorgeschlagenen Verfahren stattfinden kann (s.u., *Anpassung des Indexierungsverfahrens*). Da der Benutzer des Browser-Editors durch die Defintion neuer Kandidaten und Stoppwörter und durch die Integration neuer Konzepte Änderungen am zugrundeliegenden Modell vornehmen kann, muss ein entsprechender Mechanismus implementiert werden, um die Änderungen an das thematische Inhaltsverzeichnis und an die Konkordanzdaten weiterzugeben. Diese interne Logik des Systems wird im Abschnitt *Inkrementelles Indexieren* (s.u.) beschrieben. Die Umsetzung des Clients mit dem spezifizierten Funktionsumfang für den *View* kann weitgehend auf der Grundlage von HMTL und Javascript sowie durch den asychronen Austausch von Daten mit dem Server realisiert werden (s.u., Abschnitt *Umsetzung der Benutzeroberfläche*). Serverseitig macht dies die Bereitstellung entsprechender Dienste notwendig, die im Abschnitt *Serverseitige Komponenten* (s.u.) beschrieben sind.

Erweiterung der Datenbank

Konzeptkandidaten und Stoppwörter sind in der EFGT-Netz-Datenbank persistent zu speichern, da sie sich über die Dauer einer Sitzung mit dem Browser-Editor hinaus auf ein spezifisches EFGT-Netz sowie auf eine spezifische Dokumentsammlung beziehen. Hierfür sind zusätzliche Tabellen erforderlich, wie:

- eine Tabelle mit den Namen `candidates`, die die sprachliche Repräsentation der einzelnen Kandidaten erfasst und ihnen eine eindeutige Identifikationsnummer und einen Typen zuweist, mit dem interaktiv definierte von automatisch erkannten Kandidaten unterschieden werden. Sie dient als Grundlage für die Erzeugung eines Lexikons für

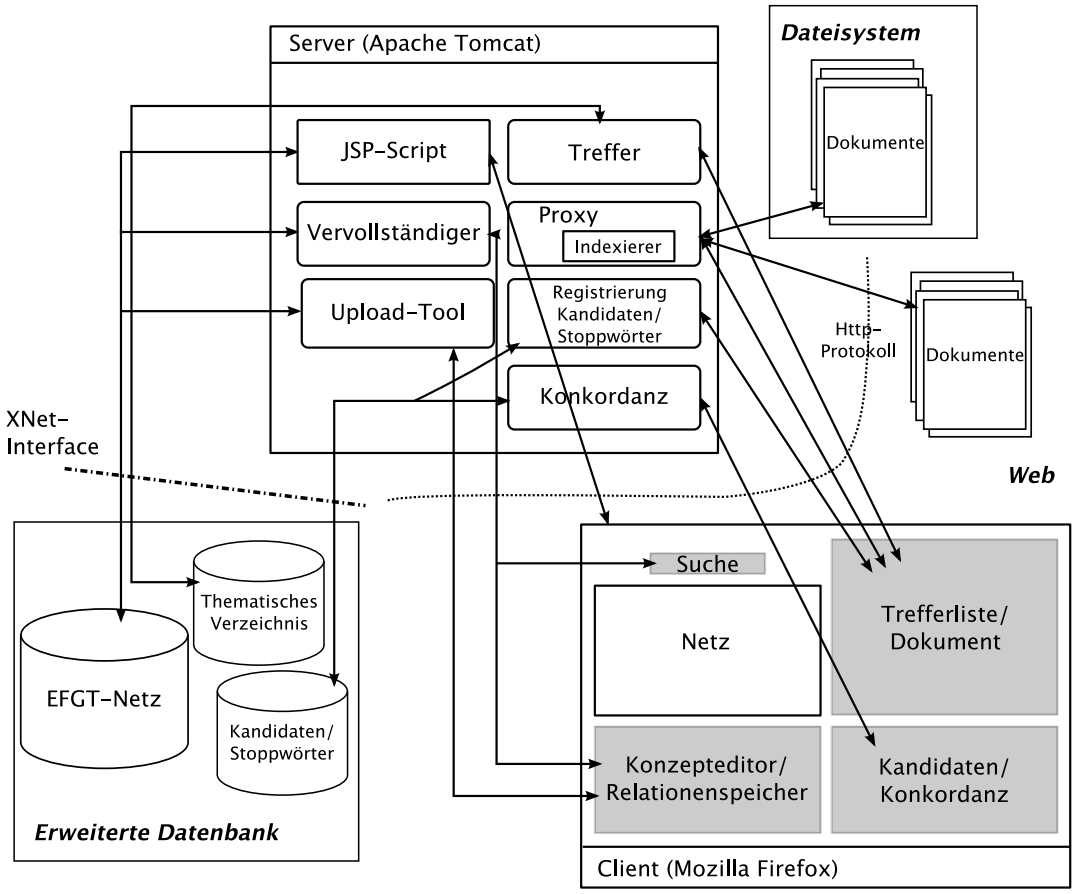


Abbildung 5.3: Eine Architektur für den Browser-Editor. Grau unterlegte Bereiche auf der Clientdarstellung kommunizieren asynchron mit dem Server; die entsprechenden Dienste sind im Server die Komponenten mit abgerundeten Ecken. Diese Architektur baut auf der des thematischen Archivbrowsers auf (vgl. Abb. 4.11, S. 121).

das inkrementelle Indexieren (s.u.). Zusätzlich dient die Tabelle `c_occurrences`, die dieselben Felder wie `occurrences` (vgl. S. 126) hat, dazu, die Vorkommen und die Frequenz der Kandidaten in den einzelnen Dokumenten zu speichern.

- entsprechende Tabellen, etwa `stopwords` und `sw_occurrences`, werden zur Verwaltung von Stoppwörtern eingesetzt. In diesem Fall ist die Speicherung von Frequenzangaben nicht nötig, da im spezifizierten Funktionsumfang der Benutzeroberfläche keine Statistiken über Stoppwörter genutzt werden.

Weiterhin werden die Tabellen gebraucht, die im Archivbrowser das thematische Inhaltsverzeichnis bilden (s. S. 126). Die Berechnung von Konkordanzen lässt sich somit durch einen *Join* der Tabellen `c_occurrences` und `occurrences` durchführen. Dokumentenlisten zu Konkordanzpaaren und spezifischen Kandidaten lassen sich ebenfalls an `c_occurrences`

ablesen.

Eine entsprechende Erweiterung des XNet-Interfaces ist für den Zugriff auf die neuen Tabellen im Server nötig.

Anpassung des Indexierungsverfahrens

Das Indexierungsprogramm muss angepasst werden, um die spezifizierten Verfahren zur Erkennung von Kandidaten (vgl. Abschnitt 5.3.2, S. 145) einzubinden und die entsprechenden Tabellen aufzubauen.

Von dieser Anpassung ist hauptsächlich der Indexierer (vgl. S. 124 sowie Abb. 4.12, S. 123) betroffen, der sich konfigurieren lassen muss, um die entsprechenden endlichen Zustandsautomaten zur Kandidatenerkennung einzubinden. Neben dem schon beim Archivbrowser vorhandenen Transduktor zur Erkennung der EFGT-Konzepte müssen im Indexierer zwei spezielle Transduktoren vorhanden sein, die jeweils für die Erkennung der vom Benutzer interaktiv definierten Kandidaten und für die Erkennung der Stoppwörter zuständig sind. Diese Transduktoren lassen sich mit demselben vorhandenen Programm (s. S. 124) erzeugen, mit dem der Konzepttransduktor aufgebaut wird. Bei der initialen Indexierung verwenden diese Transduktoren jeweils ein leeres Lexikon, das im Laufe der Interaktion mit den Benutzern aktualisiert und beim inkrementellen Indexieren (s.u.) eingesetzt wird. Abb. 5.4 zeigt eine schematische Darstellung des modifizierten Indexierers und der darin enthaltenen Komponenten.

Das beim Archivbrowser bereits verfolgte Prinzip (s. S. 121) soll in diesem Fall auch beibehalten werden, sodass Indexierungsprogramm und Proxy auf demselben Indexierer aufbauen, um die Dokumentdarstellung konsistent mit dem Index zu halten. Aus diesem Grund muss der Indexierer sowohl bei der initialen als auch bei der inkrementellen Indexierung stets dieselben Erkennungskomponenten einbinden.

Die mit dem modifizierten Indexierer erkannten Kandidaten müssen zusätzlich in den Dokumentprofilen (s. S. 122 ff) ausgegeben werden, um sie dann in der Datenbank in die Tabelle `candidates` und deren Vorkommen in `c_occurrences` einspielen zu können. Dies macht eine kleinere Anpassung des Indexierungsprogramms bei der Durchführung des fünften Schrittes nötig (vgl. Abb. 4.12, S 123).

Inkrementelles Indexieren

Die vom Benutzer interaktiv definierten Konzeptkandidaten und Stoppwörter beziehen sich nicht nur auf das Dokument, auf dem sie markiert wurden, sondern gelten für die gesamte Dokumentsammlung. Dementsprechend sind sie in den anderen Dokumenten ebenfalls darzustellen und in die Kandidatenliste und die Konkordanz aufzunehmen. Ähnlich müssen die Änderungen, die sich aus der interaktiven Integration von Konzepten im EFGT-Netz ergeben, an das thematische Inhaltsverzeichnis weitergegeben und bei der Dokumentdarstellung kenntlich gemacht werden.

Das inkrementelle Indexieren dient dazu, das Modell konsistent zu halten, wenn der Benutzer neue Elemente interaktiv definiert. Damit werden die entsprechenden Änderungen

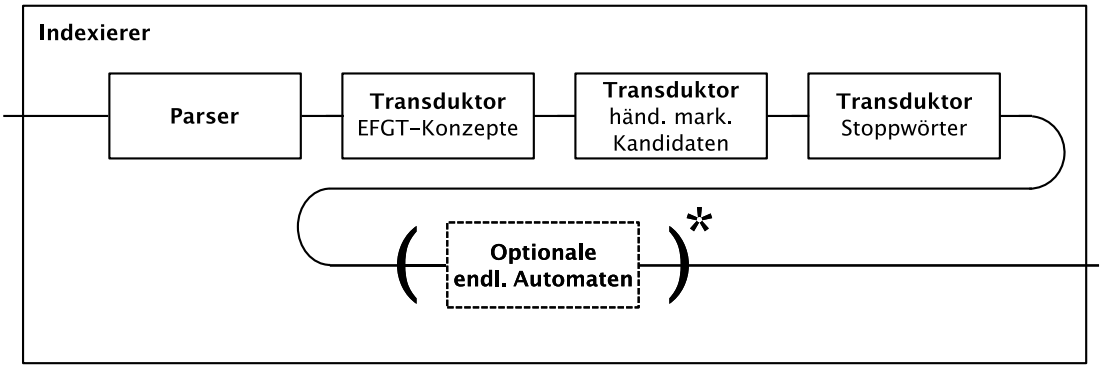


Abbildung 5.4: Interner Aufbau des Indexierers

an den Vorkommentabellen in der Datenbank vorgenommen. Die Implementierung des inkrementellen Indexierens erfolgt als zusätzlicher Ausführungsmodus des angepassten Indexierungsprogramms. In diesem Modus werden zunächst die Transduktoren aktualisiert, die vom Indexierer eingesetzt werden, um die vom Benutzer definierten Kandidaten und Stoppwörter in der gesamten Dokumentsammlung zu erkennen. Hierfür werden die zugrundeliegenden Automaten aus den Lexika der Tabelle `stopwords` bzw. `candidates` kompiliert. Wurden zudem in der Zwischenzeit dem EFGT-Netz neue Konzepte hinzugefügt, so muss ebenfalls der entsprechende Transduktor mit dem Konzeptlexikon aktualisiert werden. Beim inkrementellen Indexieren soll das Indexierungsprogramm in den Dokumentenprofilen nur die neuen erkannten Konzepte und Kandidaten ausgeben – daher inkrementell –, die dann der bestehenden Datenbank hinzugefügt werden. Der Rest des Indexierungsverfahrens erfolgt wie gewohnt.

Der zeitliche Aufwand für das inkrementelle Indexieren der gesamten Dokumentsammlung lässt sich mit dem Aufwand für die normale Indexierung abschätzen. Der maßgebliche Zeitfaktor hierbei sind die über den XNet-Interface abgewickelten, dokumentbezogenen Datenbanktransaktionen. In der aktuellen Implementierung muss man von durchschnittlich einer Sekunde pro Dokument ausgehen. Dies wirft bei größeren Dokumentsammlungen die Frage auf, ob es sinnvoll ist, bei jeder Änderung des Modells das inkrementelle Indexieren anzustoßen. Eine Alternative wäre, dies in regelmäßigen Abständen durchzuführen und bei jeder Änderung nur die Transduktorenlexika zu aktualisieren. Da das Proxy ebenfalls auf den Indexierer zugreift, könnten dadurch die Änderungen zumindest in der Darstellung der Dokumente sofort an den Benutzer weitergegeben werden. Diese Kopplung der internen Indexierung an den Zustand der Oberfläche ist in einem kollaborativen Szenario besonders relevant, da damit ein Ontologieentwickler verfolgen kann, was ein anderer mit Hilfe des Browser-Editors in derselben Dokumentensammlung bearbeitet.

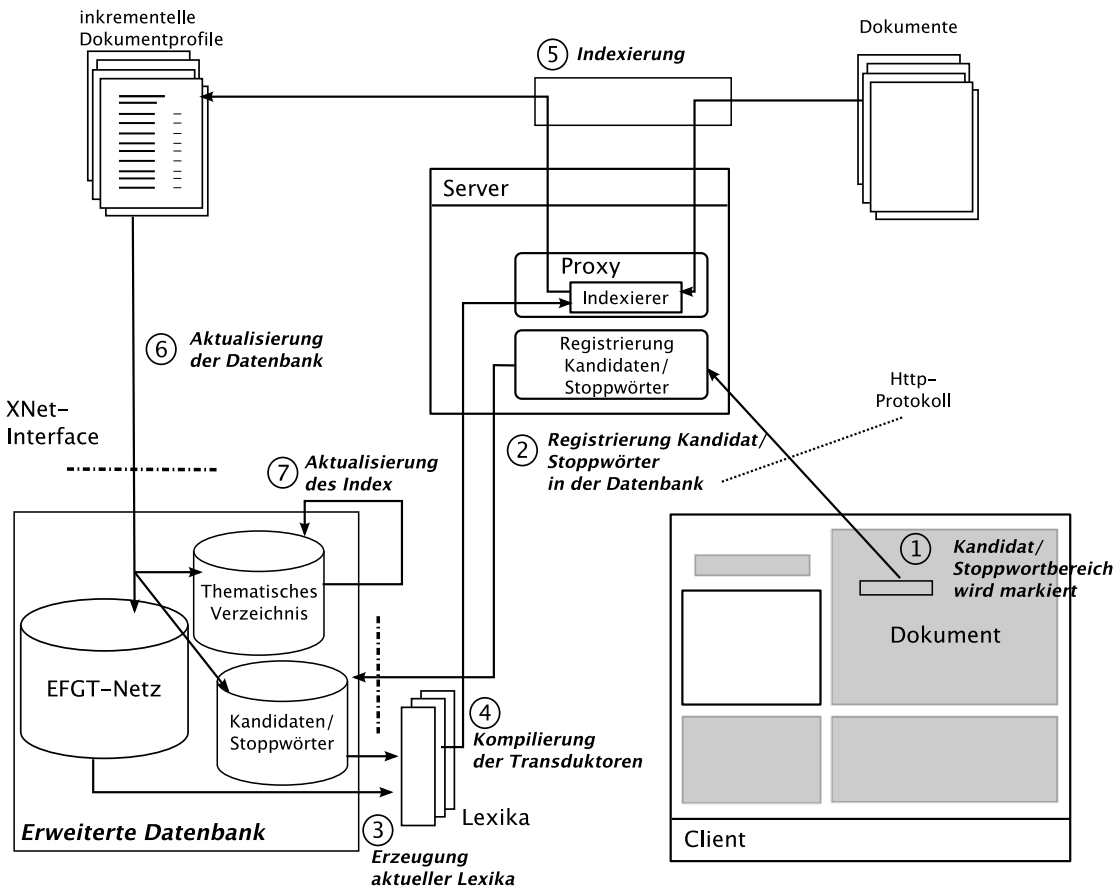


Abbildung 5.5: Vorgänge beim inkrementellen Indexieren. Im Server sind nur die beteiligten Komponenten dargestellt.

Umsetzung der Benutzeroberfläche

Für den Aufbau des *View* im Client ist das JSP-Skript, das beim Archivbrowser die Rolle des *Controllers* übernommen hat, so zu ergänzen, dass neben dem thematischen Inhaltsverzeichnis die unteren Bereiche in Abb. 5.1 dargestellt werden. Die spezifische Funktionalität des Browser-Editors baut auf asynchronen Anfragen an entsprechende Dienste im Server auf (AJAX). Dies gilt sowohl für den Konzepteditor, bei dem sich auf das Upload-Tool zurückgreifen lässt, als auch für die Kandidatenliste und die Konkordanzfunktion, die entsprechende Abfragen der (erweiterten) EFGT-Netz-Datenbank darstellen.

Die Implementierung des grundlegenden Funktionsprinzips der Oberfläche, nach dem sich Konzepte und Kandidaten mit der Maus kopieren und ziehen lassen, kann mit Hilfe einer der verschiedenen öffentlich verfügbaren Javascript (JS)-Bibliotheken durchgeführt werden, die einen Drag-and-Drop Mechanismus bereitstellen[4]. Die interaktive Markierung

[4]Das bekannteste Beispiel hiervon ist script.aculo.us (vgl. Fuchs, 2008). Ein Überblick über verschiedene

von Textpassagen als Kandidaten bzw. Stoppwörter erfordert zusätzlich die Implementierung mittels JS des angedachten Kontextmenus sowie der graphischen Auszeichnung des markierten Bereichs. Im Server muss ein entsprechender Dienst verfügbar sein, der die in der Textpassage enthaltenen Wörter entgegenimmt und in der Datenbank ablegt. Dieser Dienst ist außerdem dafür zuständig, die Suche nach weiteren Vorkommen des händisch definierten Kandidaten im Rest der Dokumentensammlung anzustoßen (inkrementelles Indexieren).

Der Mechanismus, mit dem die Eingabezeile im Konzepteditor und die durch den Relationenspeicher dargestellte Datei graphisch editiert werden können, ist ebenfalls clientseitig auf der Grundlage von HTML, JS und CSS zu implementieren.

Da die meisten Operationen durch AJAX-basierte Funktionen implementiert werden, lässt sich in den meisten Fällen ein Wiederaufbau der Clientdarstellung im Webbrowser vermeiden. In der Situation jedoch, in der ein Kandidat mit Hilfe des Konzepteditors dem EFGT-Netz hinzugefügt wurde, muss eine Änderung an mehrere, aktuell angezeigte Elemente der Oberfläche – etwa die Liste der Kandidaten, ein dargestelltes Dokument, usw. – weitergegeben werden. In diesem und ähnlichen Fällen soll der Einfachheit halber die Clientdarstellung neu aufgebaut werden, um den serverseitigen Zustand der Anwendung an die Oberfläche weiterzugeben. Das Festhalten des Oberflächenzustands vor der Aktualisierung kann durch entsprechende Cookies im Webbrowser realisiert werden. Eine Einschränkung der Benutzbarkeit der Oberfläche durch den punktuellen Neuaufbau der Darstellung ist nicht zu erwarten.

Serverseitige Komponenten

Als Gegenpart der AJAX-basierten Funktionen auf der Benutzeroberfläche müssen serverseitig eine Reihe von Diensten vorhanden sein, wobei manche übernommen werden können:

- Das *Upload-Tool* als Implementierung des Konzepteditors: Da das Upload-Tool selbst auf einer AJAX-Infrastruktur aufbaut (s. Kap 3, S. 84 ff), können die meisten seiner serverseitigen Komponenten ohne größere Modifikationen übernommen werden. Durch den vergleichsweise eingeschränkten Funktionsumfang des Konzepteditors stellt dieser Dienst eine Vereinfachung des Upload-Tools dar. Dieser Dienst muss jedoch zusätzliche Aufgaben übernehmen: Nach der Integration eines Konzeptes ins EFGT-Netz muss der ursprüngliche Kandidat aus der entsprechenden Tabelle gelöscht werden; die Aktualisierung des Konzeptlexikons und ggf. das inkrementelle Indexieren müssen angestoßen werden.

- Der Abfragedienst für die Trefferlisten des thematischen Inhaltsverzeichnisses kann vom thematischen Archivbrowser übernommen werden.

- Die Autovervollständigungsfunktion für das Suchfeld des Archivbrowsers lässt sich ebenso übernehmen. Diese Funktion kann zusätzlich für den Auswahlmechanismus

Alternativen findet sich bei ajaxPatterns.org (vgl. Mahemoff, 2008).

(s. S. 144) eingesetzt werden, der im Konzepteditor und im Relationenspeicher zur Definition von Werten in typisierten Felder dient.

- Dienste zur Abfrage der Kandidatenliste und der Konkordanzfunktion sowie der zugehörigen Dokumentlisten sind auf der Grundlage des XNet-Interfaces neu zu implementieren.

- Ein Dienst zur Verarbeitung von händisch markierten Textbereichen muss ebenfalls neu implementiert werden. Dieser Dienst ist dafür zuständig, die empfangenen Textbereiche gemäß ihres Typen (Stoppwort bzw. Kandidat) in der Datenbank abzulegen, die Aktualisierung der entsprechenden Lexika anzustoßen sowie ggf. das inkrementelle Indexieren zu starten. Kandidaten werden ohne weitere Modifikation in der `candidates`-Tabelle gespeichert. Dagegen werden Passagen mit Stoppwörtern zunächst in Einzelwörter zerlegt, um nur die noch nicht erfassten der Tabelle `stopwords` hinzuzufügen.

- Der Dienst zur Darstellung der Dokumente und Konzepttreffer, d.h. die Proxy-Komponente des Archivbrowsers, muss dahingehend erweitert werden, dass die vom neuen Indexierer erkannten Stoppwörter und Kandidaten dargestellt werden.

Der Zugriff dieser Dienste auf die erweiterte EFGT-Netz-Datenbank erfolgt über das XNet-Interface.

5.4 Schlussfolgerungen

Der in diesem Kapitel vorgestellte Browser-Editor zielt darauf ab, die Möglichkeiten, die der Archivbrowser zur Navigation in einer Dokumentensammlung bietet, für die Akquisition neuer Konzepte zu nutzen. Der Bezug, der beim Archivbrowser im thematischen Inhaltsverzeichnis zwischen EFGT-Netz und Dokumenten entsteht, ermöglicht – so die These – die thematische Einordnung der während der Navigation beobachteten, signifikativen Textbausteine. Mit der im Browser-Editor eingebauten Editierfunktion soll eine textbasierte, "navigierende" Integration neuer Konzepte in das EFGT-Netz in einer einzigen Anwendung ermöglicht werden. Dabei lässt sich das Finden signifikativer Textbausteine, die auf ein neues Konzept hinweisen, durch die Einbindung anerkannter Erkennungsverfahren unterstützen und effizient gestalten.

Mit einem als Browser-Editor ausgebauten Archivbrowser können in einer Methodologie zum Aufbau von EFGT-Netzen Texte als Quelle für die Population und Vervollständigung der Ontologie genutzt werden. Dies wird insbesondere als Beitrag zur Maintenance aufgefasst, da in vielen Bereichen Texte als Quelle von Aktualität dienen. Für das CoGE-Netz bedeutet dies beispielsweise, dessen Qualität langfristig zu sichern, indem regelmäßig Sammlungen aktueller Texte zusammengestellt und anschließend im Browser-Editor eingesetzt werden, um das Netz zu pflegen. Einzelne im CoGE-Netz erfasste Wissensbereiche lassen sich mit Hilfe entsprechender, spezialisierter Korpora ausbauen und verfeinern.

Im Lebenszyklus eines EFGT-Netzes lässt sich der Browser-Editor als komplementäre Maßnahme zum Upload-Tool auffassen: Die mit dem Upload-Tool integrierten Entitäten aus semistrukturierten Daten dienen als thematische Verankerung für Texte, die im Browser-Editor zusätzliche Kandidaten liefern.

Inwieweit der Browser-Editor für die angesprochenen Ziele nützlich ist, wird sich erst durch seine tatsächliche Verwendung zeigen. Ein kritischer Aspekt ist die Akquisition von Relationen, die als Grundlage für die Kodierung der neu gewonnenen Kandidaten dienen. Ob die Konkordanzfunktion oder die gezielte Sichtung von Kontexten ausreicht, um diese Relationen effizient zu gewinnen, lässt sich ohne praktische Tests schwer voraussagen. Selbst wenn diese Art der Wissensakquisition nicht im gewünschten Ausmaß gelingt, ist davon auszugehen, dass die Erfassung signifikativer Textbausteine und zumindest die thematische Einordnung der Kandidaten im Browser-Editor durchgeführt werden kann.

Auch wenn der Browser-Editor zunächst als Konzeption vorgestellt wurde, belegt die detaillierte Beschreibung der nötigen Schritte für die Implementierung dessen technische Machbarkeit. Dies wird dadurch untermauert, dass dabei auf bereits existierende oder im Rahmen dieser Arbeit umgesetzte entwickelte technische Infrastruktur zurückgegriffen wird, vor allem auf den Archivbrowser und das Upload-Tool.

Auch wenn im Browser-Editor sich verschiedene Erkennungsverfahren auswählen und einsetzen lassen, ist mit der hier entworfenen Konzeption nicht die Bereitstellung einer allgemeinen Toolbox intendiert, mit der sich wie bei GATE (vgl. Bontcheva et al., 2004) oder UIMA (vgl. IBM, 2007) nach Bedarf verschiedene Methoden der Textanalyse kombinieren lassen. Vielmehr wird verfolgt, die spezifischen Vorteile des thematischen Archivbrowsers für die Konzeptakquisition auszunutzen. Es bedeutet nicht, dass der Browser-Editor sich weiter ausbauen lässt, bspw. um tiefgreifendere Methoden zur linguistischen Akquisition von Kandidaten und Varianten zu berücksichtigen oder in Hinsicht auf Gewinnung von Relationen mit statistischen Mitteln. Darüber hinaus ist denkbar, im Browser-Editor auf zusätzliche Anforderungen wie die Behandlung von Ambiguitäten einzugehen, bei denen ein Bezug auf einen Korpus nützlich ist.

Ist der Browser-Editor einmal implementiert, können die damit gesammelten Erfahrungen dazu dienen, ihn zu einer Anwendung zum kollaborativen Ontologieaufbau durch nicht spezialisierte Benutzer weiterzuentwickeln. Hierfür ist vor allem eine Vereinfachung des auf dem Upload-Tool basierenden Editors notwendig, etwa durch die Bereitstellung verschiedener Schablonen.

Kapitel 6

Schlussfolgerungen und Ausblick

Allgemeines Ziel der vorliegenden Arbeit war, die beim Aufbau von EFGT-Netzen verfolgte Methodologie technisch zu unterstützen und somit dazu beizutragen, den Lebenszyklus eines EFGT-Netzes effizienter zu gestalten. Für die Definition konkreter technischer Ziele wurden in Kap. 2 zunächst die in der Praxis verfolgte Vorgehensweise und die ursprünglich vorhandene technische Infrastruktur näher in Augenschein genommen. Als Ergänzung der Infrastruktur wurden in den darauffolgenden Kapiteln das Upload-Tool, der thematische Archivbrowser und der Browser-Editor ausgearbeitet. Die ersten beiden Systeme wurden tatsächlich implementiert; die Machbarkeit des Browser-Editors wurde durch die genaue Beschreibung der nötigen, auf tatsächlich vorhandener Infrastruktur aufbauenden Implementierungsschritte belegt.

Die Bedeutung der ausgearbeiteten Maßnahmen für den Lebenszyklus eines EFGT-Netzes wird in der schematischen Darstellung in Abb. 6.1 verdeutlicht. Im Vergleich zum ursprünglichen Bild (s. Abb. 2.5, S. 49) ist darin die Rolle des Upload-Tools für die systematische Entwicklung und *Population* eines EFGT-Netzes mit Hilfe strukturierter Daten erkennbar. Damit ist in der Methodologie eine spezifische Unterstützung für die effiziente Integration von beispielsweise großen semantischen Klassen wie Personen, von vielzähligen, miteinander zusammenhängenden Entitäten wie geographischen Daten und von Teilhierarchien gegeben.

Mit dem thematischen Archivbrowser wurde eine innovative Anwendung entwickelt, die auch unter kommerziellen Aspekten interessante Merkmale aufweist. Zugleich wurde damit der Relevanz entsprochen, die eine Anwendung als konkreter Testrahmen für die Qualitätskontrolle und Weiterentwicklung der Ontologie hat – eine Erkenntnis aus der allgemeinen Analyse verschiedener Methodologien in Kap. 1.

Mit dem Browser-Editor wurden Textdokumente als weitere Quelle für die Akquisition neuer Konzepte berücksichtigt. Dadurch lässt sich zum einen das EFGT-Netz auf der Grundlage geeigneter Korpora regelmäßig ergänzen und aktualisieren (Maintenance). Zum anderen wird dadurch die Entwicklung einer auf eine spezifische Dokumentensammlung abgestimmte Ressource unterstützt (Adaption).

Indem sie verschiedene Datenquellen berücksichtigen und zu verschieden Phasen des Lebenszyklus' beitragen, beinhalten die ausgearbeiteten Programme sich gegenseitig ergän-

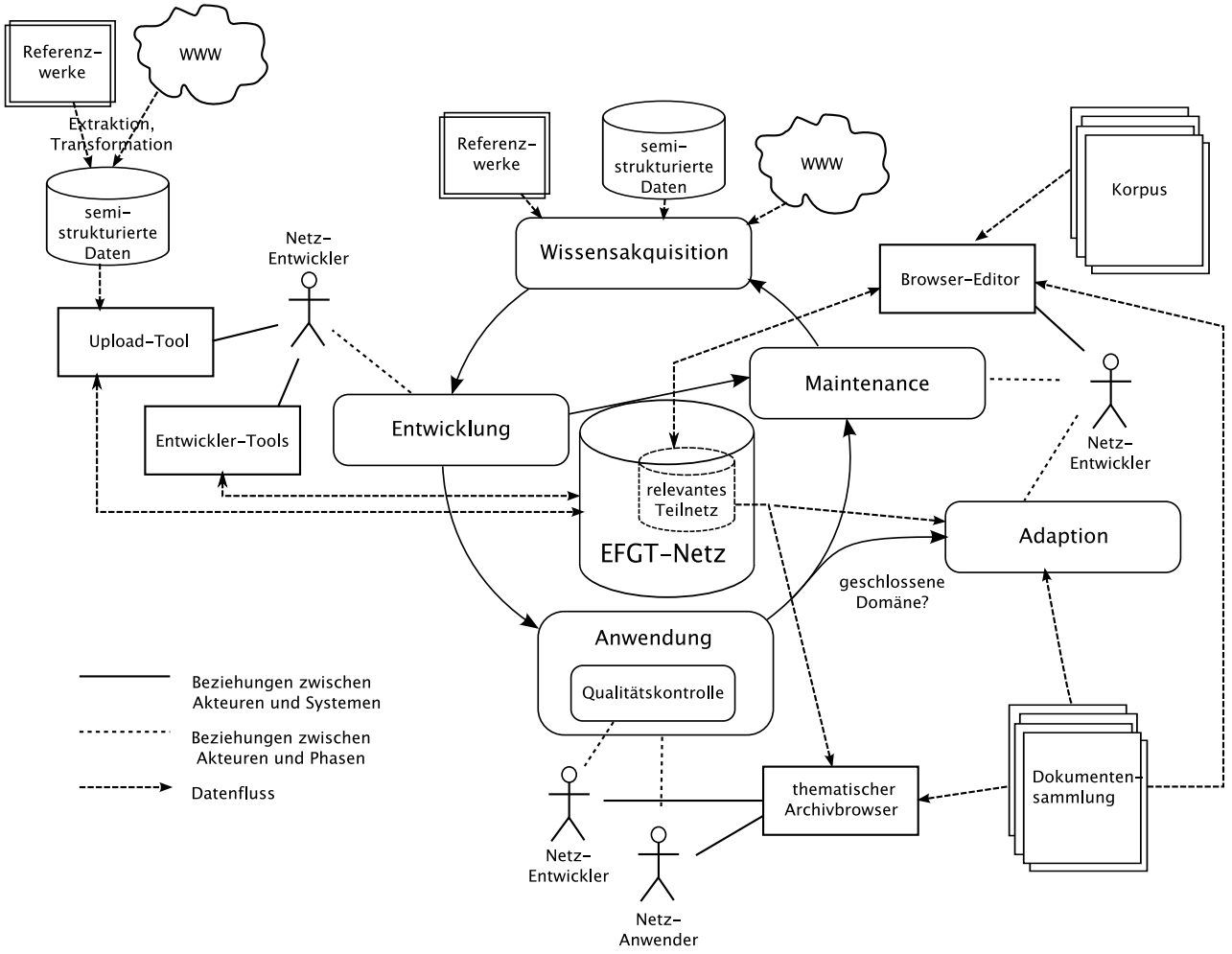


Abbildung 6.1: Einsatz der entwickelten Programme im Lebenszyklus eines EFGT-Netzes (vgl. Abb. 2.5, S. 49)

zende Maßnahmen. So dienen in einer Art Bootstrapping-Verfahren die mit dem Upload-Tool hochgeladenen Entitäten als Verankerung für die Texte, mit denen sich im Browser-Editor die Ontologie weiter verfeinern lässt. Die neu aufgenommenen Konzepte erleichtern dann eine fokussierte Akquisition weiterer Konzepte. Eine gewisse Flexibilität im Einsatz der entwickelten Programme, die eine Anpassung an unterschiedliche Projektbedingungen ermöglichen, ist ebenfalls gegeben. Beispielsweise lässt sich der Browser-Editor neben den angesprochenen Phasen der Maintenance und der Adaption auch in der Phase der Entwicklung eines Netzes mit Hilfe eines repräsentativen Korpus' einsetzen.

Die Erfahrungen, die in der Praxis bei der Entwicklung des CoGE-Netzes mit dem Upload-Tool und dem thematischen Archivbrowser gemacht werden konnten, belegen eine wesentliche Verbesserung der betroffenen Phasen des Lebenszyklus' eines EFGT-Netzes. Der systematische Ausbau des CoGE-Netzes hätte ohne das Upload-Tool kaum in dem beschriebenen Ausmaß erfolgen können. Eine angemessene Abdeckung der geographischen Achse mit dem benötigten Detailgrad wäre nicht ohne ein Hilfsmittel zur automatischen Generierung von Einträgen realisierbar gewesen. Die Visualisierung im Upload-Tool zeigte sich als geeignetes Mittel, trotz großer Datenmengen eine effiziente Überprüfung der generierten Einträge durchzuführen. Insgesamt konnte die Entwicklung des CoGE-Netzes fokussierter und effizienter ablaufen, da mit dem thematischen Archivbrowser eine für den angedachten Einsatz von EFGT-Netzen repräsentative Anwendung zur Verfügung stand. Aufgrund des Bezugs auf eine Dokumentsammlung hin ermöglichte dies, Lücken in der Abdeckung einzelner Wissensbereiche oder Probleme in der Modellierung zu entdecken. Dadurch konnte parallel zur Entwicklung eine regelmäßige Qualitätskontrolle des CoGE-Netzes stattfinden. Die Bedeutung des thematischen Archivbrowsers als einleitende Maßnahme zur Adaption des CoGE-Netzes an eine spezifische Dokumentsammlung wurde durch die Durchführung eines exemplarischen Projekts gezeigt.

Ein Resultat der Betrachtung unterschiedlicher Methodologien und Gestaltungsmöglichkeiten für den Ontologielebenszyklus in Kap. 1 war die Beobachtung, dass sich eine Verbesserung des gesamten Entwicklungsablaufs mehr durch die technische Unterstützung leicht operationalisierbarer, jedoch wichtiger Maßnahmen erreichen lässt als durch die Entwicklung zwar für sich genommen sehr effizienter, aber komplexer und schwer beherrschbarer Methoden für einzelne Aktivitäten. In dieser Hinsicht wurde mit der entwickelten Templatesprache ein prinzipieller Mechanismus zugrunde gelegt, der für verschiedene Zwecke Anwendung findet, so zur Automatisierung der Integration einzelner Datenquellen, als interne Repräsentationssprache eines graphischen Eintragseditors oder zur Definition spezifischer Formulare.

Wie am Beispiel des Browser-Editors deutlich zu erkennen ist, lassen sich grundsätzlich die bisher entwickelten Komponenten je nach Anwendungsszenario unterschiedlich miteinander kombinieren. Somit ergibt sie eine Art Toolbox, mit der sich flexibel auf Anforderungen unterschiedlicher Projektbedingungen eingehen lässt. Um dies zur vollen Geltung zu bringen ist eine weitere Modularisierung der bisher bereitgestellten Komponenten nötig, z.B. indem sie als Web-Services angeboten werden. Der Aufwand, die jetzige Implementierung in eine solche Form zu überführen, ist als relativ gering einzuschätzen, da in der auf dem AJAX-Muster aufbauenden vorhandenen Architektur die meisten Komponenten

ohnehin bereits als Dienst umgesetzt wurden. Dass die unterschiedlichen Komponenten nicht von vornherein als unabhängige Module entworfen wurden, liegt an der Ausrichtung dieser Arbeit, die stark von der Betrachtung der Anforderungen abhängt, die sich in der Praxis beim Aufbau von EFGT-Netzen stellen, und weniger vom Ziel der Entwicklung einer generischen Systemarchitektur für das *Ontology Engineering*.

Mit den vorgestellten Maßnahmen sind bei weitem nicht alle Möglichkeiten ausgeschöpft, den Lebenszyklus von EFGT-Netzen technisch zu unterstützen. So ermöglichen die vorgestellten Programme bisher zwar ein verteiltes Arbeiten, bieten aber keine zusätzliche spezifische Unterstützung für die kollaborative Ontologieentwicklung, etwa bezüglich der Erstellung von Dokumentation oder den Austausch zwischen Ontologieentwicklern. An dieser Stelle erscheint ein verstärkter Einsatz von Methoden zum *version* bzw. *change management* angebracht, um unterschiedliche Versionen der Ontologie zu verwalten, Änderungen zurückzuverfolgen oder parallele Änderungsvorschläge zu vergleichen. Es ist denkbar, dass diese Art von Methoden ebenfalls zur Vereinfachung einer sich wiederholenden Adaption an unterschiedliche Anwendungen beitragen kann. Sie ließe sich außerdem durch spezifische Methoden zur Behandlung von Ambiguitäten oder zur Anpassung der Struktur des Netzes unterstützen. In diesem Zusammenhang wäre eine interessante Fragestellung, wie sich Konzepte oder Relationen, die aus einer anwendungsunabhängigen Perspektive als einem Wissensbereich zugehörig zu betrachten sind, von denen unterscheiden lassen, die nur anwendungspezifisch relevant erscheinen. Als Ausgangspunkt hierfür könnte die Erstellung repräsentativer Korpora zu einzelnen Wissensbereichen dienen; für die Zusammenstellung solcher Korpora ist wiederum eine Weiterentwicklung des thematischen Archivbrowsers denkbar.

Abgesehen von der Optimierung von Entwicklungsabläufen bietet die praktische Anwendung von EFGT-Netzen ein zusätzliches, weites Betätigungsfeld. Neben den angesprochenen Verbesserungen im Bereich semantisches Information Retrieval wie Rankingmechanismen für den thematischen Archivbrowser, konjunktive Anfragen an eine Dokumentsammlung, thematische Kategorisierungsfunktionen, usw. können darunter sogar neue Anwendungsfelder wie automatische Zusammenfassung und Textklassifikation fallen.

Auch wenn sich der vielbeschworene *ontology bottleneck* nicht gänzlich umgehen lässt, sollte an dieser Arbeit ersichtlich geworden sein, dass sich selbst für linguistische Ontologien wie EFGT-Netze eine ganze Reihe sinnvoller Maßnahmen finden lassen, die diesen "Flaschenhals der Ontologieentwicklung" leichter passierbar machen.

Anhang A

Konkrete Syntax der EFGT-Netz-Eintragsschemata

Folgende Grammatik für *Schema*-Ausdrücke bildet die konkrete Syntax für die abstrakte Sprache $\mathbb{IDS}\text{tring}^{gen}$ aus Kap. 3, S. 66 ff.

Schema	::= *IDStringSchema* | *IfThenElse*	
IDStringSchema	::= *MarkedLocalIntro* | *MarkedSum*	
IfThenElse	::= `if` *Condition* `then` *Schema* (`else` *Schema*)? `endif`	
MarkedLocalIntro	::= `(_{` *Varname* `}` *Type Component* `.n)`	
MarkedSum	::= `(_{` *Varname* `}` *Component* (`&` *Component*)+ `)`	
Component	::= *PointerQuery*	
	| *VariableQuerySum*	
	| *DataFunction*	
	| *UnmarkedLocalIntro*	
	| *UnmarkedSum*	
	| *MarkedLocalIntro*	
	| *MarkedSum*	
PointerQuery	::= `[` *SingleQuery* `]`	
VariableQuerySum	::= `&*[` *Query* (`|` *Query*)* `]`	
DataFunction	::= `(` `IDSTR` `(` *SinglePointer* `)` `)`	// Interpretiert Belegung für *SinglePointer* als gültigen ID-String //
UnmarkedLocalIntro	::= `(` *Type Component* ? (`.` *Index*)+ `)`	
UnmarkedSum	::= `(` *Component* (`&` *Component*)+ `)`	
VariableQuerySum	::= `&*[` *Query* (`|` *Query*)* `]`	
Query	::= *SingleQuery* | *StarQuery*	
SingleQuery	::= (*SinglePointer* | *Literal*)+	
SinglePointer	::= `#` *Varname* | `#`*XPathExpression*	// Eine *XPathExpression* entspricht einem XPath-Ausdruck (vgl. XPath, 2006) //

Varname	::= [a-Z][a-Z0-9]+	// Alphanumerischer String, der mit einem Buchstaben anfängt //
Literal	::= .+	// Beliebiger String //
StarQuery	::= *Literal*? *StarPointer Literal*?	
StarPointer	::= `#*` *Varname*	
Condition	::= *SinglePointer* (	
	(*Operator Value*) |	// Vergleich laut Operator //
	(`=~ /\S/`)	// Test auf nicht-leerem String //
	) (`&&` *Condition*)*	
Operator	::= `eq` | `ne`	// Gleichheit bzw. Verschiedenheit //
Value	::= `"` *SingleQuery* `"`	
Type	::= `e` | `E` | `F` | `g` | `G` | `t` | `T`	

Literaturverzeichnis

Apache Software Foundation (2006). Apache lucene. `http://lucene.apache.org/java/docs/index.html`. Zuletzt aufgerufen am 03.12.2008.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Boston, MA, USA.

Bateman, J. A., Henschel, R., and Rinaldi, F. (1995). Generalized Upper Model 2.0: Documentation. Technical report, GMD/Institut für Integrierte Publikations- und Informationssysteme, Darmstadt, Germany.

Bernaras, A., Laresgoiti, I., and J., C. (1996). Building and reusing ontologies for electrical network applications. In Wahlser, W., editor, *European Conference on Artificial Intelligence (ECAI'96)*, pages 298–302, Budapest, Hungary. John Wiley and Sons.

Berners-Lee, T. (1999). *Weaving the Web: The Original Design and Ultimate Destinity of the World Wide Web*. HarperCollins Publishers, New York.

BibSonomy (2008). `http://www.bibsonomy.org`.

Bontas, E. P. and Tempich, C. (2005). How Much Does It Cost? Applying ONTOCOM to DILIGENT. Technical Report TR-B-05-20, Universität Karlsruhe.

Bontas, E. P. and Tempich, C. (2006). Ontology Engineering: A Reality Check. In Meersman, R., Tari, Z., et al., editors, *The 5th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE2006)*, volume 4275 of *Lecture Notes in Computer Science (LNCS)*, pages 836–854, Montpellier, France. Springer.

Bontcheva, K., Tablan, V., Maynard, D., and Cunningham, H. (2004). Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Engineering*, 10(3/4):349—373.

Borst, W. (1997). Construction of Engineering Ontologies. Technical report, Centre for Telematica and Information Technology, University of Tweenty, Enschede, The Netherlands.

Brank, J., Grobelnik, M., and Mladenić, D. (2005). A survey of ontology evaluation techniques. In *SIKDD 2005 at multiconference IS 2005*, Ljubljana, Slovenia.

Braun, S., Schmidt, A., and Walter, A. (2007). Ontology Maturing: a Collaborative Web 2.0 Approach to Ontology Engineering. In *Workshop on Collaborative Construction of Structured Knowledge (CKC) at the 16th International World Wide Web Conference (WWW07)*, Banff, Canada.

Brewster, C., Alani, H., Dasmahapatra, S., and Wilks, Y. (2004). Data Driven Ontology Evaluation. In *International Conference of Language Resources and Evaluation*, Lisbon, Portugal.

Brunner, L., Schulz, K. U., and Weigel, F. (2006). Organizing Thematic, Geographic, and Temporal Knowledge in a Well-Founded Navigation Space: Logical and Algorithmic Foundations for EFGT Nets. *Int. J. Web Service Res.*, 3(4):1–31.

Brunner, T. L. (2008). *Aufbau und Verwendung großer EFGT-Netze*. PhD thesis, Centrum für Informations- und Sprachverarbeitung der Ludwig-Maximiliams-Universität zu München, München.

Buitelaar, P. (2007). NLP in the Ontology Life-Cyle. `http://www.lt4el.eu/content/files/ws_prague/eLearning-Prague.final.pdf`. Invited Talk at eLearning4NLP Workshop, Prague.

Buitelaar, P., Declerck, T., Frank, A., Racioppa, S., Kiesel, M., Sintek, M., Engel, R., Romanelli, M., Sonntag, D., Loos, B., Micelli, V., and Cimiano, R. P. P. (2006). LingInfo: Design and Applications of a Model for the Integration of Linguistic Information in Ontologies. In *Proceedings of the OntoLex Workshop at LREC*, pages 28–32. ELRA.

Buitelaar, P., Eigner, T., and Velten, M. (2009). OntoSelect. `http://olp.dfki.de/ontoselect/`. Zuletzt aufgerufen am 26.01.2009.

Cervantes Saavedra, M. d. (1985). *Don Quijote*. Verlag Neues Leben Berlin, Berlin.

Cimiano, P. (2006). *Ontology Learning and Population from Text*. Springer.

Cimiano, P., Haase, P., Herold, M., Mantel, M., and Buitelaar, P. (2007). LexOnto: A Model for Ontology Lexicons for Ontology-based NLP. In *Proceedings of the OntoLex07 Workshop held in conjunction with ISWC'07*.

Ciorăscu, C., Ciorăscu, I., and Stoffel, K. (2003). knOWLer - Ontological Support for Information Retrieval Systems. In *Proceedings of 26th Annual International ACM SIGIR Conference, Workshop on Semantic Web*, Toronto, Canada.

De Leenheer, P. and Mens, T. (2008). Ontology Evolution: State of the Art and Future Directions. In Hepp, M., de Leenherr, P., de Moor, A., and Sure, Y., editors, *Ontology Management: Semantic Web, Semantic Web Services, and Business Applications*. Springer.

del.icio.us (2008). `http://del.icio.us`.

DOM (2005). Document object model (dom). `http://www.w3.org/DOM/`.

DWR (2008). Direct Web Remoting. `http://directwebremoting.org/`. Zuletzt aufgerufen am 10.10.2008.

Euzenat, J. (1995). Building consensual knowledge bases: context and architecture. In Mars, N., editor, *Building and sharing large knowledge bases*, pages 143–155. IOS Press, Amsterdam (NL).

Euzenat, J. and Shvaiko, P. (2007). *Ontology Matching*. Springer-Verlag, Berlin Heidelberg.

Farquhar, A., Fikes, R., and Rice, J. (1996). The ontolingua server: a tool for collaborative ontology construction. In *International Journal of Human-Computer Studies*.

Farrar and Langendoen (2003). General Ontology for Linguistic Description. `http://www.linguistics-ontology.org/gold.html`. Zuletzt aufgerufen am 02.02.2008.

Fellbaum, C., editor (1998). *WordNet. An Electronic Lexical Database*. The MIT Press, Cambridge, Mass.

Fernández-López, M. and Gómez-Pérez, A. (2002). Overview and analysis of methodologies for building ontologies. *The Knowledge Engineering Review*, 12(2):129–156.

Fernández-López, M., Gómez-Pérez, A., and Juristo, N. (1997). METHONTOLOGY: From Ontological Art Towards Ontological Engineering. In *Spring Symposium on Ontological Engineering of AAAI*, pages 33–40, California. Stanford University.

Flickr (2008). `http://www.flickr.com`.

Fowler, M. (2005). The New Methodology. `http://www.martinfowler.com/articles/newMethodology.html`. Zuletzt aufgerufen am 26.01.2009.

Fuchs, T. (2008). script.aculo.us Web 2.0 JavaScript. `http://script.aculo.us/`. Zuletzt aufgerufen am 03.01.2009.

Gross, M. (1997). The Construction of Local Grammars. In Roche, E. and Schabes, Y., editors, *Finite-State Language Processing*, pages 329–352. The MIT Press, Cambridge, Mass./London.

Gruber, T. (1993). A translation approach to portable ontology specification. *Knowledge Acquisition*, 5(2):199–220.

Gruber, T. (1995). Towards principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5/6):907–928.

Gruber, T. (2007). Folksonomy of Ontologies: A Mash-up of Apples and Oranges. *International Journal on Semantic Web and Information Systems*, 3(2). Originally published to the web 2005.

Grüninger, M. and Fox, M. S. (1995). Methodology for the design and evaluation of ontologies. In Skuce, D., editor, *IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing*, Montreal, Canada.

Grüninger, M. and Lee, J. (2002). Ontology applications and design. *Communications of the ACM*, 45(2):39–41.

Gómez-Pérez, A., Fernández-Lopez, M., and Corcho, O. (2003). *Ontological Engineering with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Advanced Information and Knowledge Processing. Springer, 1st ed. edition.

Hartmann, J., Spyns, P., Giboin, A., Maynard, D., Cuel, R., Suárez-Figueroa, M. C., and Sure, Y. (2005). Methods for Ontology Evaluation. Technical report, Knowledge Web project. Deliverable D1.2.3.

Hepp, M., de Leenherr, P., de Moor, A., and Sure, Y., editors (2008). *Ontology Management: Semantic Web, Semantic Web Services, and Business Applications*. Springer.

Herre, H., Burek, B. H. P., Hoehndorf, R., Loebe, F., and Michalek, H. (2007). General Formal Ontology (GFO): A Foundational Ontology Integrating Objects and Processes. Part I: Basic Principles. Research Group Ontologies in Medicine (OntoMed). Version 1.0.1. `http://www.onto-med.de/Archiv/ontomed2002/en/theories/gfo/part1-drafts/gfo-part1-v1-0-1.pdf`.

Huang, X.-x. and Zhou, C.-l. (2007). An OWL-based WordNet lexical ontology. *Journal of Zhejiang University - Science A*, 8(6):864–870.

IBM (2007). Unstructured Information Management Architecture (UIMA). `http://domino.research.ibm.com/comm/research_projects.nsf/pages/uima.index.html`. Zuletzt aufgerufen am 10.01.2009.

IEEE (1996). IEEE Standard for Developing Software Life Cycle Processes. Technical report, IEEE Computer Society, New York.

Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *Unified Software Development Process: The complete guide to the Unified Process from the original designers*. Addison-Wesley Longman, Amsterdam.

Kalbach, J. (2008). *Handbuch der Webnavigation*. O'Reilly Verlag, Köln.

Karapiperis, S. and Apostolous, D. (2006). Consensus Building in Collaborative Ontology Engineering Processes. *Journal Of Universal Knowledge Management*, 1(3):199–216.

Kendall, K. E. and Kendall, J. E. (1995). *Systems Analysis and Design*. Pentrice Hall, New Jersey, 3rd edition.

Lassila, O. and McGuinness, D. L. (2001). The Role of Frame-Based Representation on the Semantic Web. KSL Tech Report KSL-01-02, Stanford University. `http://www-ksl.stanford.edu/people/dlm/etai/lassila-mcguinness-fbr-sw.html`. Zuletzt aufgerufen am 20.01.2009.

Lenat, D. and Guha, R. (1990). *Building Large Knwoledge-based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, Boston, Massachusetts.

Li et al. (2004). Swoogle: A Search and Metadata Engine for the Semantic Web. In *Proceedings of the Thirteenth ACM Conference on Information and Knowledge Management*. `http://swoogle.umbc.edu/`. Zuletzt aufgerufen am 26.01.2009.

Lixto Software GmbH (2009). Lixto Visual Developer. `http://www.lixto.com/summary_products/`. Zuletzt aufgerufen am 20.01.08.

Luczak-Rösch, M. and Heese, R. (2008). Managing Ontology Lifecycles in Corporate Settings. In *Proceedings of I-SEMANTICS '08*, Graz, Austria.

Maedche, A. and Staab, S. (2001). Ontology Learning for the Semantic Web. *IEEE Intelligent Systems*, 16(2):72–79.

Maedche, A. and Volz, R. (2001). The Ontology Extraction and Maintenance Framework Text-To-Onto. In *Proceedings of the ICDM'01 Workshop on Integrating Data Mining and Knowledge Management*.

Magnini, B. and Speranza, M. (2002). Merging Global and Specialized Linguistic Ontologies. In *Proceedings of the Workshop Ontolex-2002 Ontologies and Lexical Knowledge Bases, LREC-2002*, pages 43–48.

Mahemoff, M. (2008). AjaxPatterns: Drag-And-Drop. `http://ajaxpatterns.org/Drag-And-Drop`. Zuletzt aufgerufen am 03.01.2009.

Mahesh, K. and Nirenburg, S. (1996). Meaning Representation for Knowledge Sharing in Practical Machine Translation. In *Proceedings of the FLAIRS-96 track on Information Interchange, Florida AI Research Symposium*.

McCracken, D. D. and Jackson, M. A. (1982). Life Cycle Concept Considered Harmful. *ACM Software Engineering Notes*, 7(2):29–32.

Mehrere Partner (2008a). Neon project. `http://www.neon-project.org`. Zuletzt aufgerufen am 03.02.2009.

Mehrere Partner (2008b). Neon toolkit. `http://neon-toolkit.org/`. Zuletzt aufgerufen am 03.02.2009.

Mihov, S. and Schulz, K. U. (2007). Efficient Dictionary-Based Text Rewriting using Subsequential Transducers. *Natural Language Engineering*, 13(04):353–381.

Mika, P. (2005). A Unified Model of Social Networks and Semantics. In *4th International Semantic Web Conference (ISWC 2005)*, Galway, Ireland.

MIPS (2007). The Functional Catalogue. `http://mips.gsf.de/projects/funcat`. Munich Information Center for Protein Sequences. Zuletzt aufgerufen am 10.10.2008.

Mohri, M. (1997). Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2):269–312.

Neches, R., Fikes, R., Finin, T., Gruber, T., Senator, T., and Swartout, W. (1991). Enabling technology for knowledege sharing. *AI Magazine*, 12(3):36–56.

Netscape (2009). Open Directory Project. `http://www.dmoz.org/`. Zuletzt aufgerufen am 26.01.2009.

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

Noy, N. F. and Musen, M. A. (2000). PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proc. of the National Conf. on Artificial Intelligence (AAAI)*, pages 450–455, Austin, TX.

Noy, N. F. and Musen, M. A. (2001). Anchor-PROMPT: Using NonLocal Context for Semantic Matching. In *Proc. of the Workshop on Ontologies and Information Sharing at the Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

Noy, N. F. and Musen, M. A. (2002). PromptDiff: a fixed-point algorithm for comparing ontology versions. In *Eighteenth national conference on Artificial Intelligence*, pages 744–750, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Paumier, S. (2003). *De la reconnaissance des formes linguistiques à l'analyse syntaxique*. PhD thesis, Université de Marne-la-Vallée.

Paumier, S. (2004). Unitex Corpus Processor. `http://www-igm.univ-mlv.fr/~unitex/`. Zuletzt aufgerufen am 14.01.2009.

Pinto, H. S., Staab, S., and Tempich, C. (2004). Diligent: Towards a fine-grained methodology for DIstributed, Loosely-controlled and evolvInG Engingeering of oNTologies. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, pages 393–397. IOS Press.

Rajpathak, D., Motta, E., and Roy, R. (2001). A Generic Task Ontology for Scheduling Applications. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI'2001)*, Nevada, Las Vegas, USA.

Schaffert, S., Bry, F., Baumeister, J., and Kiesel, M. (2007). Informatik Lexikon: Semantic Wiki. `http://www.gi-ev.de/no_cache/service/informatiklexikon/informatiklexikon-detailansicht/meldung/semantic-wiki-174.html`. Zuletzt aufgerufen am 02.02.2009.

Schmitz, C., Hotho, A., Jäschke, R., and Stumme, G. (2006). Kollaboratives Wissenmanagement. In Pellegrini, T. and Blumauer, A., editors, *Semantic Web: Wege zur vernetzten Wissensgesellschaft*. Springer.

Schulz, K. U. and Weigel, F. (2003). Systematics and Architecture for a Ressource Representing Knowledge about Named Entities. In Maluszynski, J., Bry, F., and Henze, N., editors, *Workshop Principles and Practice of Semantic Web Reasoning*, pages 189–207. Springer Verlag.

Simperl, E. and Sure, Y. (2008). The business view: Ontology engineering costs. In Hepp, M., de Leenherr, P., de Moor, A., and Sure, Y., editors, *Ontology Management: Semantic Web, Semantic Web Services, and Business Applications*. Springer.

Smith et al. (2007). The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology*, 25:1251 – 1255. `http://www.obofoundry.org/`. Zuletzt aufgerufen am 24.01.2009.

SPIEGEL ONLINE GmbH (2008). `http://wissen.spiegel.de/wissen/start/home.html`. Zuletzt aufgerufen am 10.12.2008.

Spyns, P. and Reinberger, M.-L. (2005). Lexically Evaluating Ontology Triples Generated Automatically from Texts. In *The Semantic Web: Research and Applications*, volume 3532/2005 of *Lecture Notes in Computer Science*, pages 563–577. Springer.

Staab, S., Schnurr, H.-P., Studer, R., and Sure, Y. (2001). Knowledge Processes and Ontologies. *IEEE Intelligent Systems*, 16(1):26–34.

Stevenson, M. (2002). Combining Disambiguation Techniques to Enrich an Ontology. In *Proceedings of the Fifteenth European Conference on Artificial Intelligence (ECAI-02), Workshop on Machine Learning and Natural Language Processing for Ontology Engineering*, Lyon, France.

Stojanovic, N., Hartmann, J., and Gonzalez, J. (2003). OntoManager – a system for usage-based ontology management. In *Proceedings of FGML Workshop*. Special Interest Group of German Information Society (FGML - Fachgruppe Maschinelles Lernen der GI e.V.).

Structural Informatics Group (2009). Foundational Model of Anatomy ontology (FMA). `http://sig.biostr.washington.edu/projects/fm/AboutFM.html`. University of Washington. Zuletzt aufgerufen am 26.01.2009.

sueddeutsche.de GmbH (2008). `http://www.sueddeutsche.de`. Zuletzt aufgerufen am 10.12.2008.

Swartout, B., Patil, R., Knight, K., and Russ, T. (1996). Toward Distributed Use of Large-Scale Ontologies. In *Proceedings of Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW96)*. `http://ksi.cpsc.ucalgary.ca/KAW/KAW96/KAW96Proc.html`.

Tanasescu, V. and Streibel, O. (2007). Extreme Tagging: Emergent Semantics through the Tagging of Tags. In *International Workshop on Emergent Semantics and Ontology Evolution at the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference*, Busan, Korea.

Teknowledge Corporation (2000). Suggested Upper Merged Ontology. `http://www.ontologyportal.org/`. Zuletzt aufgerufen am 26.01.2009.

The Gene Ontology Consortium (2000). Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29.

Torres Schumann, E., Brunner, L., Schulz, K. U., and Ringlstetter, C. (2008). A Semantic Interface for Post Secondary Education Programs. Poster bei ASIS&T 2008 Annual Meeting. Columbus, Ohio, USA.

Tran, T., Haase, P., Lewen, H., Muñoz-García, O., Gómez-Pérez, A., and Studer, R. (2007). Lifecycle-Support in Architectures for Ontology-Based Information Systems. In *Proceedings of the 6th International Semantic Web Conference (ISWC'07)*, pages 508–522, Busan, Korea.

UDC Consortium (2008). Universal Decimal Classification. `http://www.udcc.org/outline/outline.htm`.

U.S. National Library of Medicine (2006). Unified Medical Language System. `http://www.nlm.nih.gov/research/umls/`. Zuletzt aufgerufen am 26.01.2009.

Uschold, M. (1996). Building Ontologies: Towards A Unified Methodology. In *Expert System '96, 16th Annual Conference of the Britisch Computer Society Specialist Group on Expert Systems*, Cambridge, UK.

Uschold, M. and Grüninger, M. (1996). Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, 11:93–136.

Uschold, M. and Grüninger, M. (2004). Ontologies and Semantics for Seamless Connectivity. *SIGMOD Record*, 33(4).

Uschold, M. and King, M. (1995). Towards a methodology for building ontologies. In Skuce, D., editor, *IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing*, pages 6.1–6.10, Montreal, Canada.

Vossen, P., editor (1998). *EuroWordNet: A Multilingual Database with Lexical Semantic Networks*. Kluwer Academic Publishers, Dordrecht.

Vouros, G. A., Kotis, K., Chalkiopoulos, C., and Lelli, N. (2007). The HCOME-3O Framework for Supporting the Collaborative Engineering of Evolving Ontologies. In *International Workshop on Emergent Semantics and Ontology Evolution at the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference*, Busan, Korea.

Vrandecic, D., Pinto, S., Tempich, C., Sure, Y., and Salamanca, T. T. I. (2005). The DILIGENT knowledge processes. *Journal of Knowledge Management*, 9(5):85–96.

W3C (2006). RDF/OWL Representation of WordNet. `http://www.w3.org/TR/wordnet-rdf/`. W3C Working Draft 19 June 2006.

W3C Consortium (2006). XSL Transformations (XSLT), Version 1.0. W3C Recommendation 16 November 1999. http://www.w3.org/TR/xslt.

Waterfeld, W., Weiten, M., and Haase, P. (2008). Ontology Management Infrastructures. In Hepp, M., de Leenherr, P., de Moor, A., and Sure, Y., editors, *Ontology Management: Semantic Web, Semantic Web Services, and Business Applications*. Springer.

Weber, N. and Buitelaar, P. (2006). Web-based ontology learning with isolde. In *Proc. of the Workshop on Web Content Mining with Human Language at the International Semantic Web Conference*, Athens GA, USA.

Weigel, F., Schulz, K. U., Brunner, L., and Torres-Schumann, E. (2006). Integrated Document Browsing and Data Acquisition for Building Large Ontologies. In *Proceedings of the 10th International Conference on Knowledge-Based & Intelligent Information & Engineering Systems (KES), Invited Session "Engineered Applications of Semantic Web" (SWEA)*.

XEROX (2009). References To Finite-State Methods in Natural Language Processing. `http://www.xrce.xerox.com/competencies/content-analysis/fsCompiler/fsrefs.html`. Zuletzt aufgerufen am 14.01.2009.

XPath (2006). XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999. `http://www.w3.org/TR/xpath`.

Lebenslauf

Name, Vorname Torres Schumann, Eduardo

Geburt 15. Juli 1973 in Sevilla (Spanien)

Hochschulsausbildung

Juni 2009 - Juni 2005	Promotion am Centrum für Informations- und Sprachverarbeitung (CIS), Ludwig-Maximilians-Universität (LMU), München
April 1999 - Juli 2003	Magisterstudium der Computerlinguistik, Informatik und Theoretischen Linguistik an der LMU München
1993 - 1995	Studium der Physik an der Universität Augsburg, Vordiplom
1991 - 1992	Studium der Physik an der Universität Sevilla
Juni 1991	Abitur in Sevilla (Spanien)

Berufserfahrung

Seit Mai 09	Leiter der Entwicklungsabteilung der TopicZoom GmbH
Januar 08 – Mai 09	Programmierer und Gründungsmitglied der TopicZoom GmbH
Januar 09 – Sept. 09	Wissenschaftlicher Mitarbeiter am CIS, LMU München
Dez. 06 - Nov. 08	Wissenschaftlicher Mitarbeiter am CIS, LMU München. Arbeit am Firmengründungsvorhaben „SemSearch“
Okt. 2005 - Nov. 06	Wissenschaftlicher Mitarbeiter an der Universität Tübingen, DFG-Projekt „Adaptive Ontologien auf extreme Auszeichnungssprachen“. Projektleiter: Prof. Uwe Mönnich, Prof. Kai-Uwe Kühnberger
Juli 2003 - Juni 2005	Entwickler bei Fa. Biomax Informatics AG, Martinsried
Feb. 2002 - Juli 2003	Werkstudent bei Fa. Biomax Informatics AG, Martinsried, im Bereich Information Retrieval in der Biomedizin
Juli 2000 - Okt. 2001	Studentische Hilfskraft am Institut für Phonetik und Sprachliche Kommunikation, LMU München
Okt. 1999 - Juni 2000	Studentische Hilfskraft am CIS, LMU München
März 1998 - April 1999	Zivildienst bei Arbeiterwohlfahrt in Augsburg
1996 -1998	Freiberufliche Tätigkeit als Dozent für Spanisch an der Volkshochschule Augsburg

Ausgburg, den 12. März 2010