Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Konzeption und Implementierung einer semantischen Suchmaschine für Topic Maps

MASTERARBEIT

Vorgelegt von Sven Windisch B. Sc.

Betreut durch Prof. Dr. Gerhard Heyer
Dr. Lutz Maicher
Dipl. Inf. Benjamin Bock

Leipzig, Oktober 2010

ZUSAMMENFASSUNG

In den vergangenen Jahren hat die Topic-Maps-Technologie eine zunehmende Bedeutung unter den Datenintegrationstechnologien gewonnen. Für die direkte Abfrage von Informationen aus einer Topic Map existiert mit der Topic-Maps-Abfragesprache TMQL ein mächtiges Werkzeug. Um diese nutzen zu können, muss der Benutzer jedoch sowohl über Kenntnisse der Abfragesprache verfügen als auch das Schema der Topic Map kennen. Deshalb wird eine Suchmaschine benötigt, mit der auch unerfahrene Benutzer die Topic-Maps-Datenbasis durchsuchen können

Nach einer Einführung in die relevanten Topic-Maps-Grundlagen werden zunächst verschiedene auf Topic-Maps-Daten spezialisierte Indexierungsalgorithmen untersucht. Einen Spezialfall stellt dabei die Indexierung virtuell zusammengeführter Topic Maps dar. Zu diesem Problem werden verschiedene Lösungsmöglichkeiten untersucht.

Auf Basis der Suchmaschinenbibliothek Lucene wird eine semantische Suchmaschine entwickelt, welche die Topic-Maps-immanenten Elemente mit expliziter als auch mit impliziter Bedeutung sowohl bei der Indexierung als auch bei der Gewichtung der Suchergebnisse nutzt.

Darüber hinaus wird ein allgemeines Modell zur Beschreibung von Topic-Maps-basierten Facetten vorgestellt. Darauf aufbauend werden Möglichkeiten der Erstellung generischer Facetten untersucht. Weiterhin wird mit Hilfe der Topic-Maps-Abfragesprache TMQL eine Methode zur Definition von domänen-spezifischen Facetten entworfen und erläutert.

Mit der prototypischen Implementierung einer Schnittstelle, mit der die entstandene Suchmaschine in Topic-Maps-basierten Webapplikationen genutzt werden kann, wird die einfache Integration der entwickelten Suchmaschine in bestehende Web-Applikationen demonstriert. Dies wird durch die Schaffung eines neuen Pakets für die Middleware RTM ermöglicht.

DANKSAGUNG

Mein erster und größter Dank gilt meinen Eltern, die mit ihrer fortwährenden Unterstützung mein Studium und die Erstellung dieser Arbeit ermöglicht haben.

Für ihre exzellente Betreuung bedanke ich mich bei Prof. Dr. Gerhard Heyer, Dr. Lutz Maicher und Benjamin Bock. Sie haben mir stets mit Rat und Unterstützung zur Seite gestanden.

Besonderer Dank gebührt meinen Kollegen vom Topic Maps Lab, allen voran Uta Schulze, Sven Krosse und Hannes Niederhausen. Unsere vielfältigen Debatten über Topic Maps, Java und Ruby haben mich stets inspiriert.

Bedanken möchte ich mich auch bei Dr. Harald Schubert und Jürgen Schuster, die mir viele wertvolle Hinweise für die Text- und Satzgestaltung gegeben haben.

INHALTSVERZEICHNIS

1 EINFÜHRUNG 1
1.1 Motivation . . . 1
1.2 Zielstellung und Aufbau der Arbeit . . . 2
2 TOPIC-MAPS-GRUNDLAGEN 4
2.1 Das Topic-Maps-Datenmodell (TMDM) . . . 5
2.1.1 Elemente des TMDM mit expliziter Bedeutung 6
2.1.2 Elemente des TMDM mit impliziter Bedeutung 7
2.1.3 Identifizierende Elemente des TMDM . . . 11
2.2 Zusammenführen von Topic Maps . . . 11
2.3 Die Topic-Maps-Abfragesprache (TMQL) . . . 12
2.4 Das Topic Maps Application Programming Interface (TMAPI) . . . 13
3 VOLLTEXTSUCHE IN TOPIC MAPS 14
3.1 Suchmaschinen . . . 14
3.1.1 Bewertung von Suchmaschinen . . . 14
3.1.2 Verfügbare Suchbibliotheken . . . 16
3.1.3 Apache Lucene . . . 17
3.1.4 Der Lucene-Volltextindex . . . 18
3.2 Indexierung von Topic-Maps-Konstrukten . . . 20
3.2.1 Konstrukt-zentriertes Indexieren . . . 21
3.2.2 Topic-zentriertes Indexieren . . . 22
3.2.3 Evaluation der Indexierungsalgorithmen . . 22
3.2.4 Von der Topic Map zum Volltextindex . . . 24
3.2.5 Inkrementelles Reindexieren . . . 25
3.2.6 Indexierung virtuell zusammengeführter Topic Maps . . . 27
3.2.7 Entfernen eines Index . . . 28
3.3 Semantische Suche . . . 28
3.3.1 Taxonomie von Suchanfragen . . . 29
3.3.2 Analyse von Suchanfragen . . . 29
3.3.3 Lucenes Querysprache . . . 30
3.3.4 Von der Suchanfrage zur Ergebnisliste . . . 32
3.3.5 Gewichtung von Suchergebnissen . . . 33
3.3.6 Suche in virtuell zusammengeführten Topic Maps . . . 34
3.3.7 Statistische Informationen . . . 35
4 FACETTIERTE SUCHE IN TOPIC MAPS 36
4.1 Der Facettenbegriff im TMDM . . . 36
4.2 Qualitätsmetriken für Facetten . . . 37
4.2.1 Balance einer Facette . . . 37
4.2.2 Kardinalität einer Facette . . . 37
4.2.3 Frequenz einer Facette . . . 38
4.3 Facetten und Topic Maps . . . 38

4.3.1 Extraktion generischer Facetten 39
4.3.2 Definition domänenspezifischer Facetten . . 40
4.3.3 Suche im Facettenindex 42
4.3.4 Facetten aktualisieren 43
5 SUCHE IN TOPIC-MAPS-GETRIEBENEN WEBPORTALEN 44
5.1 JRuby Topic Maps 45
5.1.1 RTM-Search 46
5.1.2 RTM-Search_Hatana 47
5.2 Maiana 47
5.2.1 Indexierung in Maiana 49
5.2.2 Die Seitensuche 49
5.2.3 Die Topic-Map-Suche 50
6 ZUSAMMENFASSUNG 51
6.1 Semantische Suche in Topic Maps 51
6.2 Facettierte Suche in Topic Maps 52
6.3 Integration der Suche in Webapplikationen 52
A TERMINOLOGIE 53
B ÜBERSICHT DER INDEXIERTEN FELDER 54
B.1 Felder für identifizierende Elemente 54
B.1.1 Das Feld si 54
B.1.2 Das Feld sl 54
B.2 Felder für Elemente mit expliziter Bedeutung . . . 54
B.2.1 Das Feld name 54
B.2.2 Das Feld occurrence 55
B.3 Felder für Elemente mit impliziter Bedeutung . . . 55
B.3.1 Das Feld type 55
B.3.2 Das Feld role 55
B.3.3 Das Feld assoc_type 55
B.3.4 Das Feld assoc_player 55
B.3.5 Das Feld is_type 56
C KONFIGURATION DER KLASSEN 57
C.1 Die Klasse TopicMapIndexer 57
C.1.1 Überschreiben eines vorhandenen Index . . 57
C.1.2 Automatisches Facettieren 57
C.1.3 Indexieren identifizierender Elemente 57
C.1.4 Sprache der Grundformreduktion 58
C.2 Die Klasse TopicMapFacetter 58
C.3 Die Klasse TopicMapSearcher 58
C.3.1 Führende Platzhalter 58
C.3.2 Automatische Suche nach Facetten 58
C.3.3 Suche nach identifizierenden Elementen . . 58
C.3.4 Maximale Anzahl von Suchergebnissen . . . 59
C.3.5 Sprache der Grundformreduktion 59

LITERATURVERZEICHNIS 60

ERKLÄRUNG 65

1 EINFÜHRUNG

1.1 MOTIVATION

In den vergangenen Jahren hat die Topic-Maps-Technologie einen festen Platz unter den Datenintegrationstechnologien eingenommen. Eine vielfältige Welt von Topic-Maps-getriebenen Applikationen existiert und wird in verschiedenen Institutionen[1] genutzt. Jede dieser Institutionen nutzt Topic-Maps-Technologie, um eine unüberschaubar große Datenbasis ihrer eigenen Domäne zu verwalten und zu repräsentieren.

Für die direkte Abfrage von Informationen aus einer Topic Map existiert mit der Topic-Maps-Abfragesprache TMQL ein mächtiges Werkzeug. Um diese nutzen zu können, muss der Benutzer jedoch sowohl über Kenntnisse der Abfragesprache verfügen als auch das Schema der Topic Map kennen. Im Allgemeinen kann dies aber nicht von den Nutzern Topic-Maps-getriebener Applikationen erwartet werden. Deshalb wird eine Abfragemöglichkeit in Form einer Suchmaschine benötigt, mit der auch unerfahrene Benutzer die Topic-Maps-Datenbasis durchsuchen können [12, 38].

Sowohl Desktopapplikationen als auch Webportale, die auf der Basis von Topic Maps arbeiten, sind häufig nur mit einer klassischen Volltextsuche ausgestattet. Viele Möglichkeiten, die sich durch den hohen Grad an Strukturiertheit der Daten und die zugrunde liegende semantische Technologie für die effiziente Bearbeitung von Suchanfragen ergeben, bleiben dadurch ungenutzt. Durch eine konsequente Ausnutzung der Datenstruktur in einer semantischen Suchmaschine[2] können jedoch in allen Aufgabenbereichen von Suchmaschinen Verbesserungen erzielt werden [22]. Bereits bei der Erstellung des Index gehen viele semantische Informationen verloren, da diese in einem klassischen Volltextindex, der nur textuelle Informationen erfasst, nicht dargestellt werden können. Zusätzlich muss eine Suchmaschine mit einer Topic-Maps-Datenbasis flexible Abfragekonstrukte ermöglichen, die es dem Nutzer erlauben, die Suchergebnisse anhand ihrer Position in der zugrunde liegenden Ontologie näher zu spezifizieren.

1 Dazu gehören u. a. die Dänische Nationalbibliothek, die Polizei von Amsterdam, die norwegische Post, die nationale Steuerbehörde der USA sowie die Europäische Weltraumagentur (ESA).

2 Eine semantische Suchmaschine verarbeitet nicht nur die textuellen Inhalte der Datenbasis, sondern auch die Bedeutung der einzelnen Informationen und ihre Relationen.

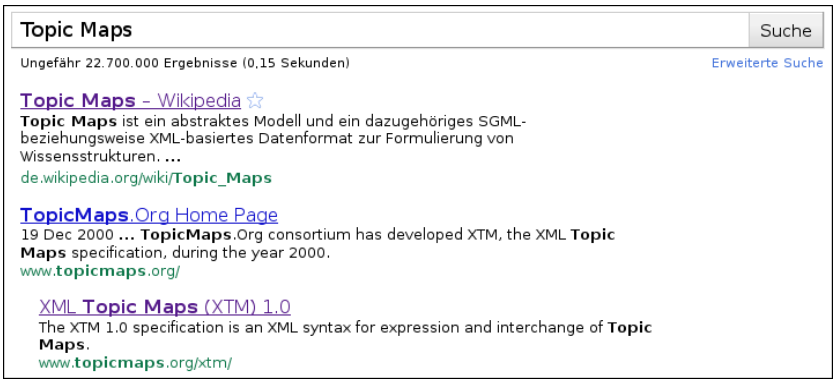

Abbildung 1: Beispiel einer klassischen Auflistung von Suchergebnissen

Für klassische Volltextsuchmaschinen hat sich ein allgemein anerkannter Standard für die Präsentation von Suchergebnissen gebildet. Dieser De-facto-Standard besteht aus dem Titel des Suchergebnisses und einem kurzen Anriss des Textes mit den gefundenen Suchwörtern (vgl. Abbildung 1). Das ist zwar für schwach strukturierte Texte geeignet, ignoriert jedoch die Möglichkeiten, die stark strukturierte Datenquellen wie Topic Maps bieten, wie etwa die Sortierung der Suchergebnisse nach dem Typ des gefundenen Topics. Eine Präsentation mittels Facetten, in denen sich die zugrunde liegende Ontologie widerspiegelt, eröffnet dem Benutzer außerdem die Möglichkeit des explorativen Navigierens durch die Suchergebnisse.

1.2 ZIELSTELLUNG UND AUFBAU DER ARBEIT

Ziel dieser Arbeit ist es, eine Suchmaschine zu entwickeln, die sowohl bei der Indexierung als auch bei der Verarbeitung von Suchanfragen die speziellen Eigenschaften von Topic Maps besonders berücksichtigt.

Kapitel 2 bietet eine Einführung in alle relevanten Topic-Maps-Technologien. Beginnend mit der Indexierung werden in Kapitel 3 alle Verarbeitungsschritte – von der Indexierung der einzelnen Topic-Maps-Konstrukte bis zur Auslieferung der gewichteten Suchergebnisse – auf die Frage hin untersucht, wie die Möglichkeiten der Topic-Maps-Technologie genutzt werden können. Da die Nutzung der Suchmaschine nicht auf eine bestimmte Topic-Maps-Engine beschränkt werden soll, wird die Suchmaschine auf einer generischen Applikationsschicht basieren.

Bevor eine Topic Map durchsucht werden kann, muss in einem geeigneten Prozess ein Index generiert werden, der sowohl die vorhandenen Informationen als auch die semantische Struktur der Topic Map berücksichtigt. Dazu wird in Kapitel 3 untersucht, wie die Struktur eines geeigneten Index aussieht und wie

die Indexierung selbst bei großen Topic Maps mit akzeptabler Geschwindigkeit durchgeführt werden kann.

Es wird untersucht, welche Kategorien von Suchanfragen in die Suchmaschine eingegeben werden können und welche Methoden den Nutzern der Suchmaschine die Anwendung erleichtern können. Da die bei der Abfrage der Topic Map ermittelten Suchergebnisse unterschiedlich relevant in Bezug auf die Suchanfrage sind, wird eine Gewichtung anhand der gegebenen Ontologieinformationen untersucht.

Neben der klassischen Volltextsuche ermöglicht die explorative Suche mit Facetten eine tiefergehende Untersuchung der Datenbasis. In Kapitel 4 werden Methoden für die Erstellung von generischen und domänenspezifischen Facetten auf der Basis einer Topic Map untersucht. Ebenso wird untersucht, anhand welcher Kriterien die generierten Facetten gewichtet werden können.

Um eine einfache Integration der semantischen Suchmaschine in neue und auch bestehende Webportale zu ermöglichen, wird für die Suchmaschine in Kapitel 5 ein Modul für die Einbindung der Bibliothek in Ruby-on-Rails-Applikationen implementiert. Weiterhin wird die prototypische Einbindung dieses Moduls innerhalb der Topic-Maps-basierten Webanwendung *Maiana* beschrieben und evaluiert.

2

TOPIC-MAPS-GRUNDLAGEN

Topic Maps dienen der Abbbildung der realen Welt in einer formalen Weise, so dass jeder Gegenstand der realen Welt, auch Aussagegegenstand genannt, durch genau ein Topic repräsentiert wird. Die Topic Map selbst beschreibt dabei keinen Aussagegegenstand, sondern dient als Container für die Topics. Diese können zwei Arten von Eigenschaften tragen:

- *Topic-Namen* sind Bezeichner in natürlicher Sprache, mit denen ein Aussagegegenstand referenziert werden kann. Für Topic-Namen können beliebig viele Varianten existieren, um den Topic-Namen beispielsweise in einer anderen Schreibweise ausdrücken zu können.

- *Belegstellen* sind typisierte[1] Eigenschaften von Topics bzw. der Aussagegegenstände, die sie repräsentieren. Die Menge der *internen Belegstellen* enthält Eigenschaften, die sich ohne externe Ressource darstellen lassen, wie beispielsweise das Geburtsdatum einer Person. Als *externe Belegstellen* werden dagegen Eigenschaften bezeichnet, die mit einer externen Ressource beschrieben werden, wie etwa die Website einer Institution.

Wie auch bei Sachverhalten in der realen Welt können vielfältige Beziehungen zwischen den einzelnen Topics bestehen. Dabei treten n Topics als Spieler von typisierten Beziehungsrollen auf, zwischen denen eine n-stellige Beziehung eines bestimmten Typs besteht [25].

Aufbauend auf diesem Topic-Maps-Paradigma hat sich in den vergangenen Jahren unter der Bezeichnung ISO/IEC 13250 eine ganze Familie von Standards entwickelt. Das Topic-Maps-Referenzmodell (TMRM) – ISO/IEC 13250-5 – dient als minimale konzeptionelle Basis für aussagegegenstandszentrierte Datenmodelle wie das TMDM und bietet eine ontologisch neutrale Terminologie, um diese zu beschreiben [16]. Das Topic-Maps-Datenmodell (TMDM) – ISO/IEC 13250-2 – definiert die abstrakte Struktur von Topic Maps sowie deren Interpretation. Zusätzlich enthält der Standard Regeln für das Zusammenführen von Topic Maps und definiert einige grundlegende indirekte Gegenstandsanzeiger [19]. Die Topic-Maps-Austauschformate CTM und XTM – ISO/IEC 13250-3 und 13260-6 – sind formal spezifizierte Austauschformate für Topic Maps. Der XTM-Standard beschreibt

1 Typisiert bedeutet in diesem Zusammenhang, dass der Wert der Belegstelle einem bestimmten Datentyp, etwa Ganzzahl oder Datum, entspricht.

eine generische XML-Notation, während der CTM-Standard die Beschreibung einer kompakteren Notation enthält [15, 17].

Neben diesen grundlegenden Standards aus der ISO/IEC 13250 Familie gibt es standardisierte Erweiterungssprachen, wie die Topic-Maps-Schemasprache (Topic Maps Constraint Language (TMCL), ISO/IEC 19756) und die Topic-Maps-Abfragesprache (Topic Maps Query Language (TMQL), ISO/IEC 18048).

Im Folgenden werden Aufbau und Datenkonstrukte des Topic-Maps-Datenmodells erläutert, da dieser Standard für diese Arbeit maßgeblich ist. Darüber hinaus wird die Topic-Maps-Abfragesprache vorgestellt. Darauf aufbauend wird in Kapitel 4 eine Möglichkeit zur Definition domänenspezifischer Topic-Facetten auf Basis von Pfadausdrücken der Topic-Maps-Abfragesprache vorgestellt.

2.1 DAS TOPIC-MAPS-DATENMODELL (TMDM)

Das Topic-Maps-Datenmodell (TMDM) besteht aus vier aufeinander aufbauenden Teilen:

1. Das *Metamodell* dient als Grundlage für die Beschreibung der Konstrukte des darauf aufbauenden Datenmodells.

2. Das *Datenmodell* bildet den umfangreichsten Teil des TMDM. In diesem Teil werden die Strukturen der einzelnen Topic-Maps-Konstrukte und ihre Zusammenhänge definiert.

3. Das *Integrationsmodell* enthält Regeln für die Identifikation der Gleichheit von Topic-Maps-Konstrukten sowie Vorschriften für das Zusammenführen dieser Konstrukte.

4. Schließlich enthält das TMDM eine Menge von *grundlegenden Gegenstandsanzeigern*, die genutzt werden können, um Interoperabilität zwischen verschiedenen Topic-Maps-basierten Applikationen zu gewährleisten.

Das Datenmodell enthält eine Menge von Topic-Map-Konstrukten, die im Folgenden beschrieben werden. Mit Blick auf die Verwendung in einer Volltextsuchmaschine lassen sich die definierten Topic-Maps-Konstrukte in drei Klassen teilen:

1. Elemente mit *explizitem Wert*, wie Belegstellen, Benennungen und Benennungsvarianten von Topics. Diese enthalten natürlichsprachige Informationen, die direkt in einen Volltextindex übernommen werden können.

2. Elemente mit *impliziter Bedeutung*, wie Topics, Topic-Typen, Beziehungen und Beziehungsrollen, enthalten dagegen semantische Informationen, die nicht direkt in einen Volltextindex übernommen werden können, sondern dafür erst ausgewertet werden müssen.

3. *Identifizierende Elemente* beschreiben Aussagegegenstände so, dass die Identität des Aussagegegenstandes eindeutig feststellbar ist.

2.1.1 *Elemente des TMDM mit expliziter Bedeutung*

2.1.1.1 *Benennungen*

Jeder Aussagegegenstand in der realen Welt verfügt über Bezeichner in natürlicher Sprache, mit denen er referenziert werden kann. Deshalb verfügt auch jedes Topic über entsprechende Topic-Namen, die aus Benennungen und den assoziierten Benennungsvarianten bestehen. Benennungen liegen immer als Zeichenkette vor. Die Menge aller Benennungen N_t eines Topics t ist folgendermaßen definiert:

$$N_t = \{n \mid n \in N^*, n \to t\} \quad , \tag{2.1}$$

wobei N^* die Menge aller Benennungen einer Topic Map umfasst. Die Schreibweise $n \to t$ bedeutet, dass die Benennung n dem Topic t zugeordnet ist, dass also das Topic mit dieser Benennung bezeichnet wird. Um verschiedene Arten von Benennungen, wie beispielsweise Vor- und Nachname einer Person, unterscheiden zu können, tragen Benennungen grundsätzlich einen Typ, der wiederum durch ein anderes Topic repräsentiert wird. Darüber hinaus verfügen Benennungen über einen Gültigkeitsbereich, der angibt, unter welchen Umständen eine Benennung zu verwenden ist. Beispielsweise ist der Name „Bundesrepublik Deutschland“ für den deutschen Staat nur gültig für den Zeitraum von 1949 bis heute.

2.1.1.2 *Benennungsvarianten*

Zusätzlich zu den Benennungen gibt es im TMDM das Konstrukt der Benennungsvarianten. Dies sind Varianten einer bestimmten Benennung, etwa um diese in verschiedenen Schreibweisen ausdrücken zu können. Dies ist beispielsweise in der japanischen Sprache notwendig, wo in den Silbenschriften Hiragana und Katakana verschiedene Schreibweisen für das gleiche Wort existieren. Die Menge aller Varianten $V_{n,t}$ zur Benennung n eines Topics t ist definiert als:

$$V_{n,t} = \{v \mid v \in V^*, v \to n, n \in N_t\} \quad , \tag{2.2}$$

wobei V^* die Menge aller Benennungsvarianten darstellt, die eine Topic Map umfasst. Die Benennungsvariante v muss darüber hinaus einer Benennung zugeordnet sein, die in der Menge der Benennungen des Topics t enthalten ist. Benennungsvarianten

haben keinen Typ, verfügen aber über ihren eigenen Gültigkeitsbereich[2]. Im Gegensatz zu den Benennungen eines Topics kann der Wert einer Benennungsvariante in jedem beliebigen Datentyp vorliegen (z. B. als Fließkommazahl oder Datum).

2.1.1.3 *Belegstellen*

Das Konstrukt der Belegstelle dient zur Verknüpfung von Aussagegegenständen mit Informationsressourcen, wobei der Aussagegegenstand durch das Topic repräsentiert wird, welches die Belegstelle enthält. Die Informationsressource kann dabei als *interne Belegstelle* einen Wert innerhalb der Topic Map darstellen oder als *externe Belegstelle* eine Ressource außerhalb der Topic Map referenzieren. Unabhängig davon ist die Menge aller Belegstellen O_t eines Topics t wie folgt definiert:

$$O_t = \{o \mid o \in O^*, o \to t\} \quad , \tag{2.3}$$

wobei O^* die Menge aller Belegstellen darstellt, die eine Topic Map enthält. Belegstellen haben grundsätzlich einen Typ. Zusätzlich wird der Wert der Belegstelle durch einen Datentyp spezifiziert. Analog zu Benennungen besitzen Belegstellen einen Gültigkeitsbereich.

2.1.2 *Elemente des TMDM mit impliziter Bedeutung*

Werden lediglich Elemente des TMDM mit expliziter Bedeutung in den Volltextindex übernommen, so wird die Suchanfrage eines Benutzers nur dann erfolgreich beantwortet werden können, wenn der Nutzer einzelne Begriffe verwendet, die im Volltextindex vorhanden sind. Sobald der Nutzer jedoch nach Begriffen sucht, die nicht explizit im Volltextindex enthalten sind, sinkt die Trefferquote (vgl. 3.1.1) der Suchanfrage rapide ab. In verschiedenen Studien konnte nachgewiesen werden, dass Benutzer von Suchmaschinen die schlechte Qualität der ihnen präsentierten Suchergebnisse nicht bemerken, da ihnen der Überblick über den gesamten Dokumentenbestand fehlt [2, 24].

Deutlich wird das Versagen des Volltextindex an folgendem Beispiel. Die Website der norwegischen Stadt Bergen ist eine Topic-Maps-basierte Webanwendung mit einer Suchmaschine, die auf einem Volltextindex basiert, der lediglich Elemente des TMDM mit expliziter Bedeutung enthält [10]. Sucht ein Nutzer

2 Da Benennungsvarianten einer Benennung fest zugeordnet sind, besteht der Gültigkeitsbereich der Benennungsvariante mindestens aus den Topics, die gemeinsam den Gültigkeitsbereich der Benennung bilden, der diese Benennungsvariante zugeordnet ist.

nach den Stichwörtern „Kindergarten“[3] und „Arna“[4], so ist die Liste der Ergebnisse unvollständig [11]. Zwar findet er alle Topics, deren Namen bzw. Belegstellen zufällig einen der beiden Begriffe enthalten. Allerdings entgehen dem Nutzer alle Topics, die zwar den Topic-Typ „Kindergarten“ tragen, dies jedoch in keiner Belegstelle explizit erwähnen. Das gleiche gilt für Topics, die über die Beziehung „liegt-in“ mit einem Topic vom Typ Adresse verbunden sind, welches wiederum Arna in einer Belegstelle anführt.

Um dieses Problem zu verstehen, ist es notwendig, die Suchintention des Nutzers in Augenschein zu nehmen. Die Intention des oben angeführten Beispielnutzers war nicht, die Vorkommen der Wörter „Kindergarten“ und „Arna“ in der Topic Map zu finden[5]. Vielmehr war der Nutzer auf der Suche nach allen Dingen vom Typ „Kindergarten“, die in irgendeiner Relation zu „Arna“ stehen. In Topic-Maps-Terminologie übersetzt, bedeutet das: Der Nutzer suchte alle Instanzen des Topic-Typs mit dem Namen „Kindergarten“, die über eine Beziehung mit der Instanz „Arna“ des Topic-Typs mit dem Namen „Ort“ verbunden sind [11].

Beziehungen enthalten keinen expliziten Informationswert an sich. Stattdessen bilden sie mit ihren Bestandteilen Rollenspieler, Beziehungsrollentyp und Beziehungstyp eine komplexe bedeutungstragende Einheit. Die explizite Bedeutung, dass ein Topic t_1 mit einem Topic t_2 verknüpft ist und dass dabei beide Topics eine definierte Rolle spielen, trägt die implizite Bedeutung einer semantischen Nähe zwischen diesen beiden Topics.

2.1.2.1 *Beziehungsrollen*

Eine Beziehungsrolle repräsentiert die Rolle, die ein Aussagegegenstand in einer Verknüpfung spielt. Eine Beziehungsrolle r kann dabei als Tupel aus dem Rollenspieler t (das Topic, welches an der Beziehung teilnimmt) und dem Typ der Beziehungsrolle $\mathrm{typ}(r)$ aufgefasst werden:

$$r = (t, \mathrm{typ}(r)) \tag{2.4}$$

Die Menge der Rollen R_t, die ein Topic t spielt, ist dann definiert als:

$$R_t = \{r \mid r \in R^*, r \to t\} \quad , \tag{2.5}$$

wobei R^* die Menge aller in der Topic Map enthaltenen Beziehungsrollen darstellt.

3 Ein norwegischer Benutzer des Stadtportals von Bergen wird natürlich nach „barnehage“, dem norwegischen Wort für Kindergarten, suchen. Die deutschen Begriffe sind hier zur besseren Verständlichkeit angegeben.

4 Arna ist ein Stadtteil von Bergen.

5 Aufgrund langjähriger Nutzung von Suchmaschinen wie Google oder Microsoft Bing, sind heute allerdings viele Nutzer auf diese Erwartungshaltung konditioniert.

2.1.2.2 *Beziehungen*

Eine Beziehung repräsentiert eine Verknüpfung zwischen mehreren Aussagegegenständen, welche eine definierte Rolle in dieser Beziehung einnehmen. Beziehungen besitzen einen Beziehungstyp, der in der Form eines Topics die Beschaffenheit der Beziehung charakterisiert. Eine Beziehung b eines bestimmten Typs typ zwischen den Beziehungsrollen von n Topics $r_{t_1}, \ldots, r_{t_n} (n \geqslant 1)$ kann dann als n-Tupel aufgefasst werden:

$$b_{typ} = (r_{t_1}, \ldots, r_{t_n}) \tag{2.6}$$

Die Menge der Beziehungen B_t, in denen ein Topic t eine Rolle spielt, ist demnach:

$$B_t = \{b \mid b \in B^*, r_t \rightarrow b\}, \tag{2.7}$$

wobei B^* die Menge aller Beziehungen umfasst, die in einer Topic Map enthalten sind, und r_t eine Beziehungsrolle aus der Menge R_t aller Beziehungsrollen des Topics t ist. Eine Übersicht über alle Konstrukte, die bei der Bildung einer binären Beziehung beteiligt sind, bietet Abbildung 2.

Einen besonderen Fall stellen *unäre Beziehungen* dar. Diese haben lediglich einen Rollenspieler und stellen daher keine Beziehung zu einem anderen Topic her. Ihre Semantik ist deshalb im Sinne einer Booleschen Variable[6] auf die Frage reduziert, ob das assoziierte Topic eine bestimmte Rolle spielt oder nicht. Beispielsweise könnte ein Dokument die Rolle eines Geheimdokumentes spielen, indem die Eigenschaft der Geheimhaltung von der Existenz einer entsprechenden unären Beziehung abhängig ist.

Alle Beziehungen verfügen über einen Gültigkeitsbereich, welcher den Kontext definiert, in dem die Verknüpfung, welche durch die Beziehung repräsentiert wird, gültig ist. Der Gültigkeitsbereich definiert darüber hinaus den Kontext, in dem die Rollenspieler ihre Beziehungsrollen erfüllen.

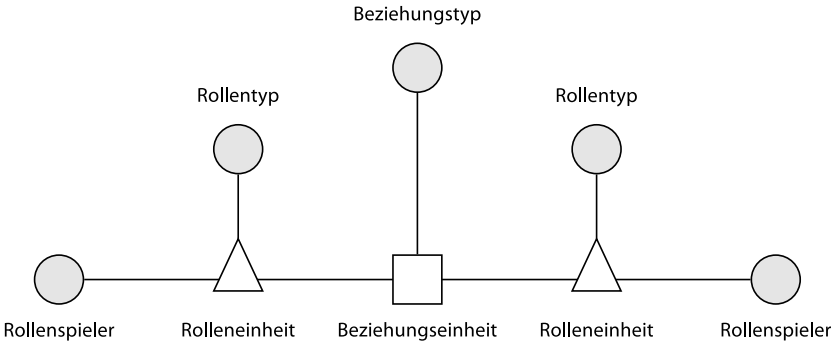

Abbildung 2: Schematische Darstellung aller Elemente des TMDM, die bei der Bildung einer (binären) Beziehung beteiligt sind.

6 Die Boolesche Variable kann dabei nur die Werte 0 (falsch) oder 1 (wahr) annehmen.

2.1.2.3 *Topic-Typen*

Topic-Typen und die Tatsache, dass Topics als Instanzen dieser Topic-Typen aufgefasst werden können, spielen eine wichtige Rolle bei der Integration von semantischen Informationen in den Volltextindex einer Topic-Maps-Suchmaschine. Ein Topic-Typ wird durch ein Topic t repräsentiert und kann folgendermaßen spezifiziert werden:

$$t \text{ ist ein Topic-Typ } \leftrightarrow \exists b_{ti}(t_i, t) \quad , \tag{2.8}$$

wobei b_{ti} eine Typ-Instanz-Beziehung und t_i ein beliebiges Topic ist, welches eine Instanz des Topic-Typs t darstellt. Die Semantik von Topic-Typen wird, wie bei allen Topics, durch die Benennungen des Topics näher bestimmt. Für den Fall, dass t den Namen „Stadt“ trägt und

$$\exists b_{ti}(t_i, t) \tag{2.9}$$

gilt, so sagt man:

„t_i ist eine Stadt.“

Wie alle Beziehungen verfügen auch Typ-Instanz-Beziehungen über einen Gültigkeitsbereich, der den Kontext ihrer Gültigkeit definiert.

2.1.2.4 *Gültigkeitsbereiche*

Für Benennungen, Benennungsvarianten, Belegstellen und Beziehungen[7] existiert im TMDM das Konstrukt des *Gültigkeitsbereiches*. Dieser repräsentiert den Kontext, in dem ein Topic-Maps-Konstrukt gültig ist. Dies bedeutet auch, dass das Topic-Maps-Konstrukt außerhalb dieses Kontextes nicht gültig ist. Formal besteht ein Gültigkeitsbereich aus mehreren Topics, die gemeinsam den Kontext definieren. Das heißt, eine Aussage ist nur dann in einem Kontext gültig, wenn alle Aussagegegenstände des Gültigkeitsbereiches Anwendung finden.

2.1.2.5 *Reifikation*

In vielen Fällen kann es hilfreich sein, zusätzliche Informationen über bestimmte Topic-Maps-Konstrukte zu speichern, z. B. um einer Beziehung einen Datumswert[8] zu geben. Dies ist jedoch nur möglich, indem ein reifizierendes Topic zur Topic Map

7 Diese vier Topic-Maps-Konstrukte werden auch unter dem Oberbegriff *Aussage* zusammengefasst.

8 Besteht beispielsweise eine Beziehung zwischen einem Topic vom Typ *Ereignis* und einem Topic vom Typ *Ort*, so kann mit einer Reifikation ein Datumswert hinzugefügt werden, der beschreibt, wann das Ereignis am verknüpften Ort geschieht.

hinzugefügt wird, welches die Eigenschaften stellvertretend für das reifizierte Topic-Maps-Konstrukt trägt. Unter *Reifikation* versteht man also das Repräsentieren des Aussagegegenstandes eines Topic-Maps-Konstruktes durch ein Topic der gleichen Topic Map[9].

2.1.3 *Identifizierende Elemente des TMDM*

Neben den Elementen mit expliziter und impliziter Information existieren im TMDM Gegenstandsanzeiger als identifizierende Elemente. Diese Elemente sind Beschreibungen des Aussagegegenstandes in Form einer Informationsressource, auf die aus einer Topic Map verwiesen wird, um einem Menschen eindeutig die Identität des Aussagegegenstandes offenzulegen. Im TMDM werden drei Klassen von identifizierenden Elementen unterschieden:

- *Indirekte Gegenstandsanzeiger* verweisen auf die Beschreibung des Aussagegegenstands. Der indirekte Gegenstandsanzeiger *http://de.wikipedia.org/wiki/Berlin* verweist beispielsweise auf die Beschreibung des Aussagegegenstands *Berlin*.
- *Direkte Gegenstandsanzeiger* verweisen auf die Informationsressource, die der Aussagegegenstand eines Topics ist. So verweist beispielsweise der direkte Gegenstandsanzeiger *http://de.wikipedia.org* auf den Aussagegegenstand *Deutsche Wikipedia*.
- Die *Identifizierung* ist die einem Konstrukt zugewiesene Adresse, welche genutzt wird, um dieses Konstrukt innerhalb der Topic Map eindeutig identifizieren zu können.

Direkte und indirekte Gegenstandsanzeiger können in beliebiger Menge verwendet werden, allerdings nur für Topics. Dagegen kann jedes Topic-Maps-Konstrukt über eine oder mehrere Identifizierungen verfügen. Aufgrund dieser Festlegungen ist die Menge der identifizierenden Elemente I_t eines Topics t definiert als:

$$I_t = \{i \mid i \in I^*, i \to t\}, \tag{2.10}$$

wobei I^* die Menge aller direkten und indirekten Gegenstandsanzeiger und Identifizierungen einer Topic Map darstellt.

2.2 ZUSAMMENFÜHREN VON TOPIC MAPS

Topic Maps ragen aus der Menge semantischer Technologien u. a. wegen ihrer effizienten Zusammenführungsmethodologie heraus,

9 Die hier eingeführte Form der Reifikation hat nichts mit dem philosophischen oder soziologischen Begriff der Reifikation gemein.

die durch standardisierte Vereinigungregeln das Zusammenführen verschiedener Topic Maps unkompliziert ermöglicht [19]. Entsprechend dem Ergebnis der Zusammenführung unterscheidet man zwei Varianten:

- Bei der *materiellen Zusammenführung* werden die zusammenzuführenden Topic Maps irreversibel miteinander verschmolzen. Die entstehende Topic Map enthält alle Elemente der einzelnen Topic Maps, wobei alle als identisch identifizierten Elemente vereinigt werden.
- Im Gegensatz dazu stellt die *virtuelle Zusammenführung* von Topic Maps eine verlustfreie Methode zur Zusammenführung dar. Dabei werden die Topic Maps nicht tatsächlich vereinigt. Vielmehr erlaubt eine zusätzliche Zusammenführungsschicht zur Anfragezeit einen Blick auf die einzelnen, scheinbar zusammengeführten Topic Maps. Dies ermöglicht später eine unkomplizierte Trennung der verschiedenen Topic Maps und erlaubt zudem die Identifizierung der Quell-Topic-Map.

2.3 DIE TOPIC-MAPS-ABFRAGESPRACHE (TMQL)

Bei der Standardisierung der Topic-Maps-Abfragesprache (TMQL) stand die Forderung nach einer MÃÂ¶glichkeit des Zugriffes auf Informationen, die gemäß dem Topic-Maps-Paradigma gespeichert sind, im Vordergrund. Ziel war die Modellierung einer effizienten und einfachen Abfragesprache, mit der komplexe Informationen und Beziehungen zwischen Topics und anderen Elementen einer Topic Map extrahiert werden können [20].

Der Standard ISO/IEC 18048 spezifiziert syntaktische Regeln für gültige Ausdrücke, die eine Extraktion von Informationen aus einer Topic Map erlauben. Weiterhin enthält der Standard Angaben über die korrekte Vorgehensweise im Abfrageprozess, im Verarbeitungsprozess und bei der Rückgabe von Ergebnissen [18].

Ein TMQL-Ausdruck kann eine von drei Ausprägungsformen annehmen: *Pfad-Ausdruck*, *Select-Ausdruck* oder *FLWR-Ausdruck*. Alle Ausprägungsformen besitzen dabei die gleiche Ausdrucksstärke, da sowohl FLWR- als auch Select-Ausdrücke eine beliebige Menge von Pfad-Ausdrücken im Kontext ihrer Klauseln verwenden können [20]. Für diese Arbeit sind die Pfad-Ausdrücke von besonderer Bedeutung, da sie die Grundlage für die Definition der in Kapitel 4 eingeführten domänenspezifischen Facetten bilden.

Ein Pfad-Ausdruck repräsentiert eine Abfolge von Navigationsschritten durch den abstrakten bidirektionalen Graphen einer Topic Map. Ausgehend von einem eindeutig identifizierbaren

Konstrukt der Topic Map können mit Navigationsschritten entlang vordefinierter Achsen einzelne Elemente aus der Topic Map extrahiert werden [18].

2.4 DAS TOPIC MAPS APPLICATION PROGRAMMING INTERFACE (TMAPI)

Das Topic Maps Application Programming Interface (TMAPI) besteht aus einer Reihe von Java-Interfaces und bildet die Grundlage vieler Topic-Maps-verarbeitender Systeme. Die erste Version wurde im Jahr 2004 veröffentlicht und wird seitdem kontinuierlich von einer aktiven Entwicklergemeinde weiterentwickelt[10]. Obwohl das TMAPI nicht von einem anerkannten Standardisierungsgremium entwickelt wurde, gilt es heute als de-facto-Standard für den Zugriff und die Manipulation von Topic Maps [13]. Neben Java ist das TMAPI mittlerweile auch für andere Sprachen verfügbar (u. a. für PHP5 [9], Ruby [4] und JRuby [3]).

Aufgrund seiner Stellung als gemeinschaftlich genutzter Pseudostandard kann das TMAPI als kleinster gemeinsamer Nenner in der heterogenen Landschaft von Topic-Maps-Applikationen gesehen werden. Für auf Topic Maps basierende Bibliotheken, wie die hier vorgestellte, ist es daher sinnvoll, direkt mit Objekten des TMAPI zu arbeiten, da die Bibliothek dann für alle TMAPI-basierten Applikationen nutzbar ist. Aus diesem Grund setzt auch die in dieser Arbeit vorgestellte semantische Suchmaschine für Topic Maps auf das TMAPI auf.

10 Die aktuelle Version 2.0.2 wurde im März 2010 veröffentlicht.

3 VOLLTEXTSUCHE IN TOPIC MAPS

3.1 SUCHMASCHINEN

Eine Suchmaschine ist ein Programm zur Recherche in einer Sammlung von Dokumenten, basierend auf einem Schlüsselwort-Index. Nach Eingabe eines Suchbegriffs liefert eine Suchmaschine eine Liste von Verweisen auf möglicherweise relevante Dokumente. Die Ausführung der verschiedenen Teilaufgaben einer Suchmaschine obliegt im Allgemeinen einem der Module, aus denen sich eine Suchmaschine zusammensetzt (vgl. Abbildung 3) [22]:

- Das *Indexierungsmodul* analysiert die Dokumente und legt den Index in einer geeigneten Datenstruktur an, indem die einzelnen Felder der Dokumente in den Index übernommen werden.
- Eingehende Suchanfragen werden vom *Abfragemodul* verarbeitet, welches die Anfrage interpretiert, den Index abfragt und die Ergebnisse sortiert und zurückliefert.
- Das *Darstellungsmodul* formt aus der Ergebnisliste eine ansprechende Darstellung. Da die Darstellung in hohem Maße von der repräsentierten Domäne abhängt, obliegt diese Aufgabe der übergeordneten Software.

3.1.1 *Bewertung von Suchmaschinen*

Obwohl die meisten Suchmaschinen in ihrer Grundstruktur übereinstimmen, sind Vergleiche zwischen ihnen schwierig. Die Qualität der Suchergebnisse hängt stark von den implementierten

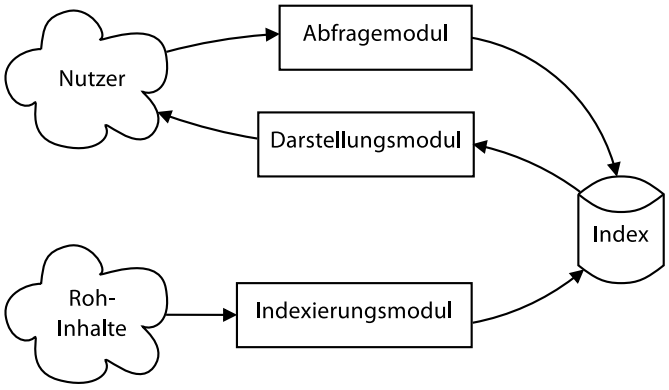

Abbildung 3: Die Komponenten einer Suchmaschine.

Indexierungs- und Suchalgorithmen und den im Index vorhandenen Daten ab.

Die einfachste Methode, die Güte einer Suchmaschine zu bestimmen, sind Relevanztests zur Bestimmung der *Trefferqualität*. Dabei werden die auf spezifische Testfragen zurückgegebenen Ergebnisse in relevante und irrelevante Ergebnisse sortiert. Für die objektive Beurteilung der Qualität einer Menge von Suchergebnissen existieren zwei Maße: *Genauigkeit* und *Trefferquote*[1]. Die Trefferquote bezeichnet die Wahrscheinlichkeit, mit der ein relevantes Dokument gefunden wird. Die Genauigkeit bestimmt die Wahrscheinlichkeit, mit der ein gefundenes Dokument für den Benutzer relevant ist. Wird für eine Ergebnismenge beispielsweise eine Trefferquote von 50 % und eine Genauigkeit von 80 % gemessen, so enthält die Ergebnismenge 50 von 100 relevanten Objekten und zusätzlich 15 irrelevante Suchergebnisse [14].

Neben der Qualität der Suchergebnisse ist bei der Bewertung einer Suchmaschine auch die Qualität des Index selbst von Bedeutung. Diese *Index-Qualität* wird über die Vollständigkeit sowie die Aktualität des Index bestimmt. Dabei gilt der Index einer Suchmaschine als umso besser, je vollständiger die Datenbasis in ihm abgebildet ist und je kürzer die Zeit ist, die vergeht, bis sich eine Änderung der Datenbasis im Volltextindex niederschlägt [24].

Als drittes Qualitätsmerkmal spielt die *Recherche-Qualität* eine Rolle. Da unterschiedliche Nutzer voneinander abweichende Vorstellungen über die Bedienungsweise einer Suchmaschine haben können, sollte eine Suchmaschine auch verschiedene Bedienungsweisen anbieten. Dies beinhaltet vor allem, dass die Anfragesprache der Suchmaschine sowohl einfache Anfragen mit wenigen Suchwörtern als auch komplexe Kombinationen mit logischen Operatoren unterstützt [23].

Ein Parameter der Recherche-Qualität ist von überragender Wichtigkeit und soll deshalb gesondert genannt werden: *Effizienz*. Darunter ist die Effizienz der eingesetzten Algorithmen und Datenstrukturen hinsichtlich ihres Verbrauchs von Speicher und der Länge ihrer Laufzeit zu verstehen.

Neben den genannten messbaren Bewertungskriterien spielt die *Nutzerzufriedenheit* eine wichtige Rolle bei der Bewertung einer Suchmaschine. Diese kann durch Nutzerbefragungen, Laboruntersuchungen[2] oder die Analyse automatisch generierter Log-Dateien erfolgen [24]. Da die Ergebnisse dieser Untersuchungen stark von der Domäne der Daten und der Präsentation der

1 Diese Maße werden in der Fachliteratur im Allgemeinen als *Recall* und *Precision* bezeichnet.

2 Dabei werden die Nutzer unter kontrollierten Bedingungen bei der Benutzung der Suchmaschine beobachtet. Diese Methode bietet wesentlich genauere Testergebnisse als allgemeine Nutzerbefragungen, ist jedoch sehr zeit- und kostenintensiv.

Suchergebnisse abhängen, kann eine solche Untersuchung nicht Teil dieser Arbeit sein.

3.1.2 *Verfügbare Suchbibliotheken*

Da der Fokus dieser Arbeit nicht auf der Entwicklung der Suchmaschine selbst liegt, sondern vielmehr auf der konkreten Anwendung für Daten, die als Topic Map vorliegen, soll eine existierende Suchmaschinenbibliothek als Grundlage dienen. Damit die Bibliothek zur Suche in Topic Maps später der Topic-Maps-Gemeinschaft zur freien Benutzung übergeben werden kann, kommen aus dem vielfältigen Angebot an verfügbaren Suchbibliotheken nur die in Frage, deren Lizenz eine solche Veröffentlichung erlaubt. Daraus ergeben sich die in Tabelle 1 aufgeführten Kandidaten.

Außerdem wird eine möglichst breite Verwendbarkeit der entwickelten Topic-Maps-Suchbibliothek angestrebt, sodass diese kompatibel mit dem Topic Maps Application Interface (TMAPI) sein muss (vgl. Abschnitt 2.4). Die Java-Implementierung des TMAPI hat bislang die weiteste Verbreitung gefunden, weshalb als Programmiersprache ebenfalls nur Java in Betracht gezogen werden kann. Mit diesen Vorgaben stehen lediglich noch die Bibliotheken *Lucene* und *Minion* als Kandidaten zur Verfügung.

Weitere wichtige Kriterien für die Auswahl der Suchbibliothek bilden die Menge der zur Verfügung stehenden Literatur sowie der Aktivitätsgrad der Entwicklergemeinde. Die Entwicklerge-

BIBLIOTHEK	SPRACHE	EINGABEDATEN
DataparkSearch	C	Spezialisiert auf Websites
Ferret	Ruby	Kein spezielles Format
ht:/Dig	C++	Spezialisiert auf Websites
Lucene	Java	Kein spezielles Format
Minion	Java	Kein spezielles Format
mnoGoSearch	C	Spezialisiert auf relationale Datenbanken und XML-Dateien
Sphinx	C++	Spezialisiert auf relationale Datenbanken und XML-Dateien
Swish-e	C	Spezialisiert auf Text-, HTML- und XML-Dateien
Xapian	C++	Kein spezielles Format

Tabelle 1: Evaluierte Suchbibliotheken

meinde von Lucene ist äußerst aktiv, während das Minion-Projekt seit dem Jahr 2009 nur noch wenig Aktivität aufweist. Darüber hinaus existieren für die Bibliothek Minion keine Nachschlagewerke, während für Lucene gleich mehrere Bücher erhältlich sind. Aufgrund dieser Erkenntnisse dient Lucene als Grundlage für die zu entwickelnde Topic-Maps-Suchmaschine.

3.1.3 *Apache Lucene*

Lucene ist keine vollständige Suchmaschine, wie sie im vorangegangenen Abschnitt 3.1 beschrieben wird. Vielmehr ist Lucene eine Softwarebibliothek, die einzelne Module und Funktionen für das Indexieren und Durchsuchen großer Datenmengen bereitstellt (vgl. Abbildung 4). Durch diese Konzentration auf einige Kernfunktionalitäten ist Lucene in der Lage, diese Funktionalitäten mit außerordentlich großer Effizienz durchzuführen [27].

Die Softwarebibliothek Lucene entstand im Jahr 2000 als Open-Source-Projekt und wird seit 2002 von der Apache Foundation betreut[3]. Zum gegenwärtigen Zeitpunkt ist Lucene in der Version 3.0.2 verfügbar. Diese Version dient auch als Grundlage dieser Arbeit.

Lucene konnte seine Stabilität bereits in vielfältigen Anwendungsszenarien unter Beweis stellen. Darunter finden sich prominente Webportale wie Twitter, MySpace oder LinkedIn genauso wie die integrierte Entwicklungsumgebung Eclipse oder der Browser Epiphany[4].

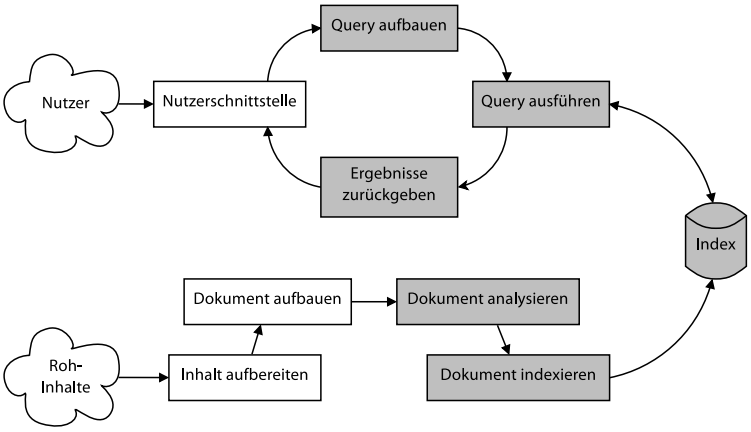

Abbildung 4: Übersicht über die von Lucene zur Verfügung gestellten Funktionen. Die *grau* hinterlegten Komponenten sind innerhalb von Lucene zu finden (nach [27]), die *weiß* hinterlegten Komponenten müssen zusätzlich implementiert werden.

3 Da Lucene ein Open-Source-Projekt ist, sind sowohl Dokumentation als auch Quelltext online unter http://lucene.apache.org verfügbar.

4 Eine umfangreiche Auflistung von Projekten, die mit Lucene realisiert wurden, findet sich unter http://wiki.apache.org/jakarta-lucene/PoweredBy.

Neben der Kernbibliothek gibt es im Paket *Lucene Contrib*[5] vielfältige Zusatzbibliotheken, die zusätzliche Funktionalitäten wie Rechtschreibprüfung oder Grundformreduktion zur Verfügung stellen.

3.1.4 *Der Lucene-Volltextindex*

Der Lucene-Volltextindex ist ein *invertierter Index*, was eine schnelle Abfrage des Index ermöglicht [27]. Ein invertierter Index besteht aus zwei Komponenten:

1. Das *Vokabular* enthält eine Liste der unterschiedlichen Wörter im indexierten Text.
2. In einer zweiten Liste wird die Position jedes Auftretens eines indexierten Wortes im indexierten Text gespeichert.

Durch diese Struktur kann für jedes Suchwort schnell bestimmt werden, ob es im Index auftritt und falls ja, in welchen Dokument das Suchwort vorkommt [1].

3.1.4.1 *Terme*

Die Basis des Lucene-Volltextindex bilden die *Terme*. Diese werden als Zuordnung zwischen einem Wort und einem Feldnamen gespeichert und enthalten darüber hinaus Informationen über die Dokumentenfrequenz df, also die Anzahl der Dokumente innerhalb eines Index, die diesen Term enthalten.

3.1.4.2 *Felder*

Eine weitere grundlegende Datenstruktur für den Aufbau des Lucene-Volltextindex bilden die *Felder*, die als Tripel $f = (n, w, g)$ gespeichert werden, wobei n den Namen des Feldes und w den Wert und g die Gewichtung des Feldes beschreibt. Jedes Dokument kann beliebig viele Felder des gleichen Namens enthalten, sodass mehrere Werte einem Feldnamen zugeordnet werden können.

Durch diese feingranulare Aufteilung des Volltextindex ist Lucene in der Lage, den Einträgen ein gewisses Maß an Semantik zuzuordnen. So können Inhalte verschiedener Herkunft in diskriminierende Felder sortiert werden. Das ermöglicht Suchanfragen mit einem gezielten Zugriff auf einzelne Inhalte des Index.

Der Wert w eines Feldes kann entweder aus Freitext oder einem atomaren Schlüsselwort bestehen. Besteht der Wert aus Freitext, so wird er vor dem Indexieren analysiert (vgl. 3.2.4). Atomare

5 Die umfangreiche Liste aller verfügbaren Zusatzbibliotheken kann unter http://lucene.apache.org/java/3_0_2/lucene-contrib/index.html abgerufen werden.

Schlüsselwörter werden dagegen ohne Analyse indexiert. Damit können Datumsangaben oder URLs indexiert werden, die sonst im Analyseprozess fälschlicherweise zerlegt würden. Indexierte Felder können zusätzlich den ursprünglichen und nichtanalysierten Wert speichern, sodass dieser bei der Suche zurückgegeben werden kann.

Diese Konfigurationsmöglichkeiten werden genutzt, um die verschiedenen indexierbaren Elemente des TMDM entsprechend ihrer Natur behandeln zu können:

- Informationen über die identifizierenden Elemente eines Topics (direkte und indirekte Gegenstandsanzeiger sowie Identifizierungen) werden in Feldern vom Typ `IdentifierField` gespeichert. Diese Felder speichern den ursprünglichen Wert, ohne ihn durch eine Analyse zu zerlegen.

- Die expliziten und impliziten Informationen eines Topics werden in Feldern des Typs `ContentField` gespeichert, die einerseits den ursprünglichen Wert, andererseits aber auch den analysierten Wert enthalten.

- Für Belegstellen mit numerischen Werten[6] existieren außerdem Felder vom Typ `NumericField`, die den ursprünglichen Wert numerisch und nicht textuell interpretieren. Damit werden Abfragen nach einer bestimmten Spanne von numerischen Werten ermöglicht (vgl. 3.3.3).

Eine umfassende Auflistung aller gespeicherten Felder findet sich in Anhang B.

3.1.4.3 *Dokumente*

Der klassische Anwendungsfall einer Volltextsuchmaschine basiert auf einer Sammlung von Dokumenten. Aus diesen Dokumenten können einzelne Felder wie *Titel* oder *Autor* extrahiert werden. Diese werden dann, zusammen mit dem Feld *Inhalt*, das den Text des Dokumentes enthält, in den Index übernommen. Damit die einzelnen Informationen bei der Beantwortung von Suchanfragen zueinander zugeordnet werden können, werden die informationstragenden Felder zu *Dokumenten* zusammengefasst.

Dies geschieht analog auch mit den einzelnen Feldern eines Topics. Wird beispielsweise ein Suchwort in einem *name*-Feld gefunden, so kann durch die Zugehörigkeit zu einem bestimmten Dokument das identifizierende *si*-Feld ermittelt werden.

6 Hiermit sind sowohl Ganzzahl- als auch Fließkommawerte gemeint.

3.1.4.4 *Textanalyse und Grundformreduktion*

Zu indexierender Text liegt im Allgemeinen in seiner Rohform vor. Das heißt, dass der Text erst in seine Bestandteile zerlegt werden muss, um die einzelnen Terme in Feldern speichern zu können. Die Standardanalyse umfasst dabei folgende Schritte:

1. Der `StandardFilter` zerlegt den Text in einzelne Wörter, wobei Leerzeichen, Zeilenumbrüche, Bindestriche und Interpunktionszeichen als Wortgrenzen erkannt werden. Eine genaue Beschreibung des Algorithmus findet sich in Abschnitt 4.3. von [27].

2. Mit dem `LowerCaseFilter` werden die Buchstaben aller Wörter in Kleinbuchstaben umgewandelt. Dadurch wird eine größere Flexibiliät für den Nutzer erreicht, der die gleichen Ergebnisse bekommt, egal ob er *bank*, *Bank* oder *BANK* in seiner Suchanfrage verwendet.

3. Im Anschluss daran entfernt der `StopFilter` alle Stoppwörter. Als Stoppwörter werden alle Wörter betrachtet, die sehr häufig in Texten auftreten, aber keine Bedeutung haben. Dazu zählen u. a. bestimmte und unbestimmte Artikel, Konjunktionen und einige Präpositionen.

Einen weiteren nützlichen Umformungsschritt stellt die *Grundformreduktion* dar. Darunter versteht man die Rückführung aller flektierten Wortformen in ihre Grundform [14]. Dies bringt weitere Flexibilität für den Nutzer der Suchmaschine, da er damit die gleichen Suchergebnisse erhält, wenn er nach *Bank* und *Bänke* sucht [31]. Die Funktionalität der Grundformreduktion wird durch das Lucene-Contrib-Paket *Snowball* bereitgestellt. Dieses Paket bietet Grundformreduktion für 19 verschiedene Sprachen[7].

Bei der Grundformreduktion ist jedoch zu beachten, dass die Regeln für die Flexion von Wörtern von Sprache zu Sprache verschieden sind. Daher müssen die Klassen `TopicMapIndexer` und `TopicMapSearcher` entprechend konfiguriert werden, damit sie den passenden Reduktionsalgorithmus ausführen. Eine Übersicht über die Konfigurationsmöglichkeiten der Klassen findet sich in Anhang C.

3.2 INDEXIERUNG VON TOPIC-MAPS-KONSTRUKTEN

Die in Abschnitt 2.1.1 vorgestellten Topic-Maps-Konstrukte unterscheiden sich hinsichtlich ihrer Bedeutung für ihr zugehöriges Topic. Deshalb ist es zweckmäßig, die verschiedenen Konstrukte auch in verschiedene Felder eines Lucene-Dokuments

7 Eine genaue Übersicht der verfügbaren Sprachen und der verwandten Algorithmen findet sich unter http://snowball.tartarus.org.

zu sortieren. Abhängig von der Betrachtungsweise existieren zwei verschiedene Möglichkeiten der Indexierung von Topic-Maps-Konstrukten. Zum einen kann man den Fokus auf das Topic-Maps-Konstrukt selbst legen, zum anderen kann das Topic als zentrales Element dienen.

3.2.1 *Konstrukt-zentriertes Indexieren*

Bei diesem Paradigma steht das Topic-Maps-Konstrukt selbst im Mittelpunkt. Für jedes in der Topic Map vorhandene Topic werden die in Abschnitt 2.1.1 spezifizierten Konstrukte einzeln indexiert. Dazu wird jedes Konstrukt als Quadrupel (I_k, k, e, w) aufgefasst, wobei I_k die Menge der identifizierenden Elemente des Topic-Maps-Konstrukts, k die Klasse[8], e die Id des Elternelements und w den eigentlichen Wert des Konstrukts darstellt. Jedes dieser Quadrupel bildet ein Dokument des Indexes, so dass jedes Dokument eine fixe Menge an Feldern aufweist.

Die folgende schematische Darstellung verdeutlicht die Arbeitsweise des Konstrukt-zentrierten Indexierungsalgorithmus:

```
for each Topic t in TopicMap
  for each Name n in t
    index(id(n), 'N', id(t), value(n))
    for each Variante v in n
      index(id(v), 'V', id(n), value(v))
    end
  end
  for each Belegstelle o in t
    index(id(o), 'O', id(t), value(o))
  end
end
```

Es wird deutlich, dass mit diesem Algorithmus lediglich die Elemente mit expliziter Bedeutung erfasst werden. Die Elemente mit impliziter Bedeutung können aber analog indexiert werden, wie der folgende Algorithmus zeigt:

```
for each Beziehung b in TopicMap
  index(id(b), 'B', id(tm), value(b))
  for each Rolle r in b
    index(id(r), 'R', id(b), value(r))
  end
end
```

Dieser Algorithmus wirft jedoch zwei Probleme auf. Zum einen ist unklar, welches Ergebnis der Aufruf der Funktionen `value(a)` und `value(r)` liefert, da Beziehungen ja gerade keinen expliziten Wert tragen, sondern durch ihre Existenz eine implizite Bedeutung haben. Zum anderen kann mit diesem Algorithmus kein Zusammenhang zwischen den in den Topic Map vorhandenen

8 Klasse beschreibt hier, ob es sich bei dem Konstrukt um eine Benennung, eine Benennungsvariante oder eine Belegstelle handelt.

Topics und ihren Beziehungen hergestellt werden. Dieser Zusammenhang muss nach der Ermittlung der Suchergebnisse erneut geknüpft werden.

3.2.2 *Topic-zentriertes Indexieren*

Im Gegensatz zum Konstrukt-zentrierten Ansatz steht bei diesem Algorithmus das Topic im Mittelpunkt. Jedes Topic wird als ein Dokument des Index aufgefasst, wobei die jeweiligen Unterkonstrukte als Felder des Dokuments dienen. Jeder Indexeintrag eines Topics t besteht aus einem 6-Tupel $(I_t, N_t, V_t, O_t, R_t, B_t)$, wobei die Typ-Instanz-Beziehungen des Topics aus der Menge B_t herausgelöst und in einem separaten Feld gespeichert werden.

```
for each Topic t in TopicMap
  N ← {n | n ∈ name(t)}
  V ← {v | v ∈ variante(t)}
  O ← {o | o ∈ belegstelle(t)}
  R ← {r | r ∈ rolle(t)}
  B ← {b | b ∈ beziehung(t)}
  index(id(t), N, V, O, R, B)
end
```

Dieser Algorithmus indexiert für jedes Topic sowohl die Konstrukte mit expliziter als auch die Konstrukte mit impliziter Bedeutung. Allerdings kann auch dieser Ansatz nicht alle in einer Topic Map enthaltenen Informationen in den Index übernehmen. Im Gegensatz zum Konstrukt-zentrierten Ansatz fehlen im Index die Identifizierungen der Benennungen und Belegstellen. Dies ist jedoch unproblematisch, da der Zweck einer Volltextsuchmaschine gerade nicht in einer Abfragemöglichkeit für URIs besteht, sondern dem Nutzer die Suche mit natürlichsprachlichen Suchbegriffen ermöglichen soll.

3.2.3 *Evaluation der Indexierungsalgorithmen*

Wie bereits in Abschnitt 3.1.1 beschrieben, zählt die Effizienz zu den wichtigsten Bewertungskriterien einer Suchmaschine. Daher steht bei der Wahl des geeigneten Indexierungsalgorithmus die Geschwindigkeit im Vordergrund. Da die in dieser Arbeit vorgestellte Suchmaschine außerdem in einem größeren Rahmen eingesetzt werden soll, ist die Effizienz bei der Verwendung von Speicherplatz für den Index von Interesse.

Um das Skalierungsverhalten der vorgestellten Indexierungsalgorithmen bestimmen zu können, wurden vier Topic Maps verschiedener Größe verwandt:

- Die **toyTM** enthält alle im Topic-Maps-Datenmodell [19] beschriebenen Konstrukte und dient daher als Referenz-Topic-Map bei der Entwicklung Topic-Maps-basierter Software.

ALG.	TOYTM	LOS	MM	C16
		Größe der Topic Map im TNO-Maß[10]		
	279	7336	9343	155274
				Indexierung (in ms)
1	200,0	723,6	734,5	5474,3
2	179,2	557,1	629,6	3961,6
				Suche (in ms)
1	61,2	68,3	59,6	69,5
2	69,1	64,3	63,3	64,5
				Speicherplatz (in KiB)
1	27,2	742,2	985,0	28400
2	17,1	553,6	635,3	19600

Tabelle 2: Ergebnisse der Evaluation der verschiedenen Indexierungsalgorithmen. Der Konstrukt-zentrierte Ansatz wird als *Alg. 1* bezeichnet, der Topic-zentrierte Ansatz als *Alg. 2*

- Die **Musica-Migrans-Topic-Map (mm)** enthält die Daten des Musica-Migrans-Portals[9].
- Die Topic Map **Los** enthält eine allgemeine Spezifizierung des öffentlichen Sektors Norwegens.
- Die aus einem internen Forschungsprojekt stammende Topic Map **C16** enthält annähernd 10000 Topics mit Informationen über Start-Up-Firmen.

Um die beiden Ansätze miteinander vergleichen zu können, wurde jeweils nur die Indexierung der Elemente mit expliziter Information durchgeführt. Die in Tabelle 2 dargestellten Ergebnisse zeigen deutlich die Vorteile des Topic-zentrierten Ansatzes. Bei der zur Indexierung benötigten Zeit ist dieser Ansatz zwischen 11 % und 28 % schneller als der Konstrukt-zentrierte Ansatz. Gleiches gilt für den vom jeweiligen Index belegten Speicherplatz. Dieser ist beim Topic-zentrierten Ansatz im Vergleich zum Konstrukt-zentrierten Ansatz von 25 % bis 37 % kleiner. Die zur Suche benötigte Zeit schwankt bei beiden Ansätzen für alle Topic Maps nur leicht. Die in Tabelle 2 angegebenen Werte liegen alle innerhalb der Standardabweichung der Messung.

9 Das Projekt Musica Migrans sammelt Daten über städtische Musikinstitutionen im Mittel- und Osteuropa des 19. und 20. Jahrhundert.

10 Das TNO-Maß ist ein einfaches Maß zur Bestimmung der Komplexität einer gegebenen Topic Map. Dabei wird die Anzahl der Topics, Topic-Namen und Belegstellen addiert.

Der Topic-zentrierte Ansatz bietet mehrere Vorteile. Zum einen bietet er eine höhere Index-Qualität, da die in den Index übernommenen Informationen die indexierte Topic Map besser widerspiegeln. Zum anderen haben die Messungen gezeigt, dass der Topic-zentrierte Ansatz auch bezüglich der Effizienz einige Vorteile bietet. Damit sorgt dieser Algorithmus dafür, dass die Suchmaschine bei zwei wesentlichen Bewertungskriterien sehr gute Werte erhält (vgl. Abschnitt 3.1.1).

3.2.4 *Von der Topic Map zum Volltextindex*

Für die Erstellung eines Volltextindex aus einem TMAPI-kompatiblen Topic-Map-Objekt wird die Klasse `TopicMapIndexer` genutzt. Bei der Initialisierung wird zunächst festgelegt, ob der Index im Hauptspeicher oder auf der Festplatte angelegt wird. Darüber hinaus kann die Klasse umfangreich konfiguriert werden (vgl. Anhang C).

Der Indexierungsvorgang wird durch einen Aufruf der Methode `index()` gestartet. Diese Methode erwartet ein Topic-Map-Objekt als Parameter und prüft zunächst, ob ein eventuell bereits vorhandener Index überschrieben werden darf.

Fällt dieser Test positiv aus, so wird für jedes Topic der gegebenen Topic Map die Methode `createTopicDocument()` der Klasse `DocumentFactory` aufgerufen. Diese Methode implementiert den in Abschnitt 3.2.2 eingeführten Algorithmus so, dass für jedes Konstrukt der Wert bestimmt und in einem entsprechenden Feld gespeichert wird. Eine Übersicht über die Herkunft der einzelnen Werte ist in Tabelle 3 zu sehen.

Zusätzlich zu den indexierten Termen erhält jedes Feld ein Basisgewicht, das später bei der Gewichtung der Suchergebnisse eine Rolle spielt. Da das endgültige Gewicht eines Suchergebnisses durch Multiplikation der verschiedenen Werte errechnet wird, sorgt ein Basisgewicht größer als $1,0$ dafür, dass Übereinstimmungen in den entsprechenden Feldern weiter oben in der Ergebnisliste stehen.

In einer klassischen Volltextsuche werden alle Felder mit dem gleichen Basisgewicht versehen, da bereits die aus der Übereinstimmung von Suchwörtern resultierende Gewichtung eine gute Beschreibung der Treffergenauigkeit abgibt. Da die Suchergebnisse jedoch die vorhandenen strukturierten Informationen widerspiegeln sollen, werden die Felder für Benennungen und Topic-Typen mit einem leicht höheren Gewicht versehen.

Nachdem ein Dokument für das zu indexierende Topic erstellt wurde, wird zusätzlich die Information über vorhandene Typ-Instanz-Beziehungen in das Feld *is_type* übernommen. Damit kann später die Ergebnisliste in Topic-Typen und Topic-Instanzen

KONSTRUKT	HERKUNFT DER WERTE	GEWICHT
Beziehungstyp	Die Namen der Typen der Beziehungen, an denen das Topic teilnimmt	1,0
Belegstelle	Der Wert der Belegstellen	1,0
Benennung	Die Namen und Namensvarianten des Topics	Namen: 1,5 Varianten: 1,0
Beziehungsrolle	Die Namen der Rollen, die das Topic spielt	1,0
Gegenstandsanzeiger	Die direkten und indirekten Gegenstandsanzeiger des Topics	1,0
Rollenspieler	Die Namen aller anderen Topics, die an einer Beziehung teilnehmen	1,0
Topic-Typ	Die Namen des Topic-Typs	2,0

Tabelle 3: Übersicht der Topic-Map-Konstrukte, ihrer Verwendung im Volltextindex und ihrer Basisgewichte

aufgeteilt werden. Zusätzlich wird die für das Indexieren benötigte Zeit aufgezeichnet und als Metadatum im Index gespeichert.

3.2.5 *Inkrementelles Reindexieren*

Um das in Abschnitt 3.1.1 aufgeführte Qualitätskriterium der hohen Index-Qualität stets zu erreichen, ist es notwendig, die Änderungen in den indexierten Topic Maps auch im Volltextindex nachzuvollziehen. Dabei ist es nicht zweckmäßig, gerade bei größeren Topic Maps, dass immer wieder die gesamte Topic Map indexiert werden muss. Stattdessen muss ein *inkrementelles Reindexieren* durchgeführt werden, bei dem stets nur die Topics neu indexiert werden, die sich tatsächlich geändert haben.

Unter inkrementellem Reindexieren versteht man die Methode, Änderungen am Index vorzunehmen, ohne die gesamte Datenbasis zu reindexieren. Im vorliegenden Fall bedeutet das, sollte sich der Wert eines einzelnen Konstruktes der Topic Map ändern, so muss lediglich das Dokument des zum Konstrukt gehörenden Topics neu erstellt werden, statt dies für alle Topics der Topic Map zu tun.

Apache Lucene bietet keine spezifische Methode für das Reindexieren eines bestimmten Dokumentes an. Stattdessen wird mit der Lucene-Suche eine Menge von zu reindexierenden Dokumenten erstellt, die zuerst gänzlich aus dem Index entfernt

werden. Danach können die betreffenden Dokumente neu indexiert werden. Diese Methode kann auch für das inkrementelle Reindexieren einzelner Topics genutzt werden.

Jedes Topic einer Topic Map wird innerhalb dieser Topic Map eindeutig durch die Menge seiner Gegenstandsanzeiger und Identifizierungen bezeichnet. Durch das Indexieren dieser Adresse ist es möglich, das Dokument des neu zu indexierenden Topics im Index zu bestimmen und anschließend gezielt zu löschen. Danach kann das einzelne Topic neu indexiert werden.

Dies geschieht mit der Methode `update()` der Klasse `TopicMapIndexer`, die das neu zu indexierende Topic-Objekt sowie den Update-Typ als Parameter erwartet. Die verfügbaren Update-Typen werden in Tabelle 4 beschrieben.

Die Information über die neu zu indexierenden Topics können über verschiedene Wege eintreffen:

- Der `update()`-Methode kann ein einzelnes, mit der TMAPI kompatibles Topic-Objekt übergeben werden. Abhängig vom gewählten Update-Typ wird der Indexeintrag des Topics bearbeitet.
- Die TMQL-Implementierung *TMQL4J* ist in der Lage, Nachrichten an andere Bibliotheken zu verschicken, sobald sich ein Topic-Map-Konstrukt ändert. Für diesen Fall kann die Klasse `EventListener` benutzt werden, um die Nachrichten von TMQL4J entgegenzunehmen und in entsprechende Aufrufe der `update()`-Methode zu übersetzen.

UPDATE-TYP	FUNKTIONSWEISE
ADD	Mit diesem Update-Typ können neue Topics zum Index hinzugefügt werden.
REMOVE	Dieser Update-Typ dient zur Entfernung von Einträgen aus dem Index.
MODIFY	Die Änderung eines vorhandenen Indexeintrages eines bestimmten Topics kann mit diesem Update-Typ durchgeführt werden, sofern der Indexeintrag bereits existiert.
FORCE_ADD	Auch mit diesem Update-Typ kann ein spezifischer Indexeintrag geändert werden. Sollte kein Indexeintrag für das Topic existieren, so wird er angelegt.

Tabelle 4: Update-Typen und ihre Bedeutung

3.2.6 *Indexierung virtuell zusammengeführter Topic Maps*

Wie bereits in Abschnitt 2.2 angesprochen, muss die Indexierung zusammengeführter Topic Maps je nach Methode der Zusammenführung, materiell oder virtuell, gesondert betrachtet werden.

Nach einer materiellen Zusammenführung steht als Ergebnis genau eine Topic Map zur Verfügung. Diese kann wie jede andere Topic Map indexiert und durchsucht werden. Im Gegensatz dazu existieren für virtuell zusammengeführte Topic Maps zwei verschiedene Indexierungsansätze:

1. Im einfachen Fall wird der `index()`-Methode der Klasse `TopicMapIndexer` das Ergebnis der virtuellen Zusammenführung als Topic-Map-Objekt übergeben. Ist dies der Fall, so existiert genau ein Index mit der Adresse der zusammengeführten Topic Map. Dieser kann dann durchsucht werden, wobei die Suchergebnisse korrekt gegen die zusammengeführte Topic Map aufgelöst werden können (vgl. Abbildung 5).

2. Der kompliziertere Fall liegt dann vor, wenn die zusammenzuführenden Topic Maps einzeln indexiert werden und erst zum späteren Suchzeitpunkt in einer virtuell zusammengeführten Topic Map vorliegen. Dann existiert kein Index mit der Adresse der zusammengeführten Topic Map, aber

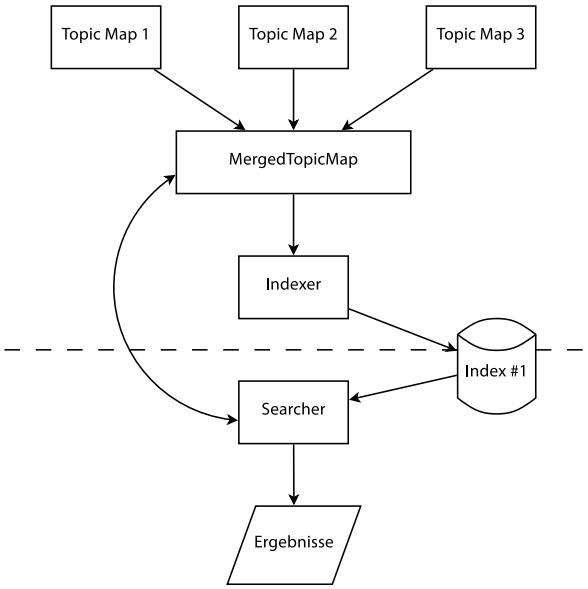

Abbildung 5: Schema der Indexierung einer a priori virtuell zusammengeführten Topic Map. Zum Indexierungszeitpunkt liegen die einzelnen Topic Maps bereits als virtuell zusammengeführte Topic Map vor. Deshalb wird nur die zusammengeführte Topic Map indexiert und kann später unkompliziert durchsucht werden

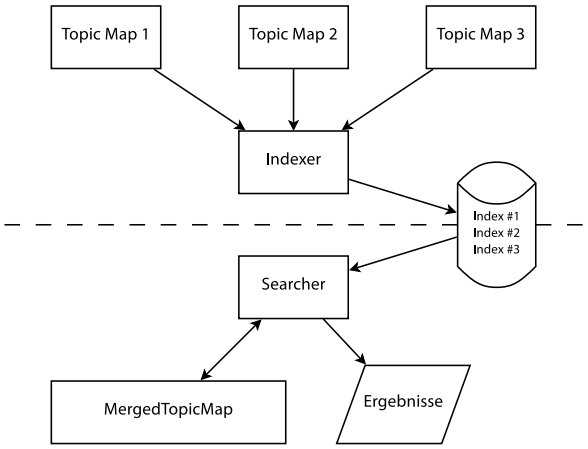

Abbildung 6: Schema der Indexierung einer nicht a priori virtuell zusammengeführte Topic Map. Die Topic Maps werden einzeln indexiert. Zum Suchzeitpunkt liegt jedoch eine virtuell zusammengeführte Topic Map vor. Die zusammengeführten Topic Maps werden identifiziert und im Zusammenspiel mit der virtuell zusammengeführten Topic Map wird die Ergebnisliste erstellt

es existieren einzelne Indexe für die repräsentierten Topic Maps (vgl. Abbildung 6). Die Suchergebnisse können gegen die virtuell zusammengeführte Topic Map aufgelöst werden. Dabei werden gegebenenfalls mehrfach auftretende Elemente aus der Ergebnisliste entfernt.

3.2.7 *Entfernen eines Index*

Neben den vielfältigen Methoden zum Anlegen oder Modifizieren eines Index existiert mit der Methode `delete()` in der Klasse `TopicMapIndexer` die Möglichkeit, einen Index zu entfernen. Diese Methode löscht die Index-Dateien und gibt nach einem erfolgreichen Löschvorgang den Wert `true` zurück. Sollte das Löschen fehlschlagen, so wird der Wert `false` zurückgegeben.

3.3 SEMANTISCHE SUCHE

Strukturierte Daten, wie beispielsweise Topic Maps, können mit verschiedenen Methoden durchsucht werden. Für die direkte Abfrage von Daten aus einer Topic Map existiert die Topic-Maps-Abfragesprache TMQL. Um diese nutzen zu können, muss der Benutzer jedoch sowohl über Kenntnisse der Abfragesprache verfügen als auch das Schema der Topic Map kennen, was im Allgemeinen nicht von den Nutzern Topic-Maps-getriebener Applikationen erwartet werden kann [12, 38].

Dieses Problem kann umgangen werden, indem dem Nutzer eine vereinfachte Abfragesprache präsentiert wird, welche vor

der Abfrage der Datenbasis in TMQL-Anweisungen übersetzt wird [21, 32]. Nutzer von Suchmaschinen sind jedoch daran gewöhnt, Suchanfragen in natürlicher Sprache stellen zu können, und nur wenige machen tatsächlich Gebrauch von den erweiterten Möglichkeiten einer Abfragesprache.

Daher ist es notwendig, die zu durchsuchende Datenbasis in einer Art zu repräsentieren, die eine einfache Anfrage mit Suchwörtern ermöglicht. Dies wird durch die Verwendung eines Volltextindex ermöglicht.

3.3.1 *Taxonomie von Suchanfragen*

An Suchmaschinen gestellte Suchanfragen bilden eine sehr heterogene Menge. Unabhängig von ihrer Komplexität oder der indexierten Domäne steht bei der Formulierung einer Suchanfrage eins der folgenden Paradigmen im Vordergrund [5]:

- *Informationsorientierte Suche* – Der Nutzer sucht bei informationalen Anfragen nach Informationen zu einem bestimmten Thema. Mit dem Erhalt der Suchergebnisliste ist die Suche beendet und der Nutzer beginnt mit der Auswertung der Suchergebnisse. Suchanfragen diesen Typs können von einer Volltextsuchmaschine beantwortet werden, wie sie in diesem Kapitel vorgestellt wird.

- *Navigationsorientierte Suche* – Der Nutzer startet bei navigationalen Anfragen mit gewissen Dokumenten der Datenbasis, die er bereits kennt. Davon ausgehend sucht der Nutzer jedoch nach weiteren Informationen im Umfeld der ihm bereits bekannten Dokumente. Diese Art von Suchanfragen kann mit einer facettierten Suche bearbeitet werden, wie sie in Kapitel 4 beschrieben wird.

- *Transaktionsorientierte Suche* – Der Nutzer sucht bei transaktionalen Anfragen nach Elementen, mit denen er weiterarbeiten möchte. Dies sind bei Websuchmaschinen zum Beispiel Internetshops oder Chats. Dieses Paradigma spielt für die hier vorgestellte Suchmaschine für Topic Maps keine Rolle.

3.3.2 *Analyse von Suchanfragen*

Obwohl Topic Maps eine abstrakte Methode zur Datenrepräsentation darstellen, sind die repräsentierten Daten im Allgemeinen natürlichsprachlich[11]. Zusätzlich unterliegen die Suchanfragen,

11 Für die Verwendung in einer Volltextsuchmaschine ist dies sogar eine elememtare Grundbedingung.

da sie von Menschen gestellt werden, einer gewissen Heterogenität. Damit ergeben sich vielfältige Probleme, die Suchmaschinen im Sinne einer möglichst angenehmen Benutzung addressieren müssen.

Ein erstes Problem stellen falsch geschriebene Wörter dar, die sowohl in der Datenbasis als auch in Suchanfragen vorkommen können. Um dieses Problem zu umgehen, existiert im Bereich des Information Retrieval eine Methode, bei der mit Hilfe eines Wörterbuches versucht wird, den oder die möglichen Schreibfehler zu finden. Dabei wird eine Reihe von einfachen Änderungen am Ursprungswort vorgenommen und nach jeder Übersetzung erneut im Wörterbuch nachgeschlagen. Das Wort, welches mit den geringsten Änderungen gefunden werden kann, wird dann als das wahrscheinlich richtige zurückgegeben [6]. Diese Methode eignet sich gut für die automatische Korrektur, wird aber aufgrund ihrer Komplexität hier nicht in Betracht gezogen.

Als weiteres Problem stellt sich die polysemische Eigenschaft vieler Wörter dar. Die Ambiguität (Mehrdeutigkeit) von Wörtern sorgt dafür, dass keine eineindeutige Zuordnung zwischen einem Wort und seiner Bedeutung hergestellt werden kann. Zum Beispiel ist es, ohne den weiteren Kontext zu kennen, nicht möglich herauszufinden, ob mit dem Wort *Bank* ein Geldinstitut oder ein Sitzmöbel gemeint ist. Mit Hilfe des Kontextes kann jedoch die gewünschte Bedeutung bestimmt werden. Dazu werden für alle Wörter signifikante Kookkurenzen ermittelt und in Cluster sortiert. Beispielsweise kann so festgestellt werden, dass mit *Bank* ein Sitzmöbel gemeint ist, wenn das Cluster der signifikanten Kookkurenzen zusätzliche Wörter, wie *Park* oder *sitzen*, enthält [14]. Da mit einer Topic Map jedoch im Allgemeinen nur eine Domäne dargestellt wird, erübrigt sich dieses Problem für Topic-Maps-basierte Suchmaschinen.

Das dritte Problem, das in allen natürlichen Sprachen vorhanden ist, entsteht aus den vielfältigen Möglichkeiten zur Flexion von Wörtern. So wird beispielsweise eine Belegstelle, die das Wort *Abendrot* enthält, nicht gefunden, wenn der Nutzer nach dem Wort *Abendröte* sucht, obwohl beide Wörter die gleiche Bedeutung tragen. Dieses Problem wird gelöst, indem bei der Indexierung und bei der Analyse von Suchanfragen eine Grundformreduktion durchgeführt wird [31]. Dadurch werden die indexierten Wörter und die Suchwörter einander angeglichen, wodurch die Trefferquote steigt. Die Konfiguration der Grundformreduktion und ihre Verwendung in Lucene wird in Abschnitt 3.1.4.4 besprochen.

3.3.3 *Lucenes Querysprache*

Um den Nutzern der Suchmaschine eine größtmögliche Flexibilität bei der Formulierung der Suchanfragen bieten zu können,

muss die Anfragesprache der Suchmaschine neben einfachen Anfragen mit Suchwörtern auch differenzierte Anfragen ermöglichen. Die zur Verfügung stehende Anfragesprache von Lucene bietet die folgenden Möglichkeiten:

- Neben vollständigen Suchwörtern können auch unvollständige Suchwörter mit Platzhaltern für die Suche verwendet werden. Als Platzhalter kann das Zeichen `?` verwendet werden, um ein Zeichen im Suchwort zu ersetzen. Das Zeichen $*$ ersetzt dagegen beliebig viele Zeichen im Suchwort. Die Platzhalter können sowohl am Ende als auch an beliebiger Stelle im Suchwort auftreten. Über die Konfigurationseinstellung `allowLeadingWildcard` ist es möglich, auch Platzhalter am Anfang von Suchwörtern zu benutzen. Da dies jedoch die Suche verlangsamt, ist der voreingestellte Wert für diese Eigenschaft `false`.

- Suchanfragen können auf allen Feldern des Index, auf einzelnen Feldern oder einer Mischung daraus ausgeführt werden. Um eine Suchanfrage auf einem einzelnen Feld auszuführen, wird dem Suchwort der Feldname vorangestellt (`feld:Suchwort`). Das Zeichen `:` dient dabei als Trennung zwischen Feldname und Suchwort. Daraus ergibt sich, dass Suchwörter, die ein `:` enthalten, in Anführungszeichen eingeschlossen werden müssen (`feld:"http://example.org"`). Suchbegriffe ohne Feldangabe werden in allen Feldern gesucht. Eine Übersicht über die durchsuchbaren Felder findet sich in Anhang B.

- Zusätzlich zur Suche nach einzelnen Werten ist auch die Suche nach Bereichen von Werten möglich. Eine Bereichssuche kann mit der Anfrage `feld:[minWert TO maxWert]` durchgeführt werden. Sollen die Werte `minWert` und `maxWert` selbst nicht mehr zur Menge der Suchergebnisse gehören, so lautet die Suchanfrage `feld:{minWert TO maxWert}`.

- Eine wichtige Eigenschaft für die Flexibilität einer Anfragesprache ist die Möglichkeit der Kombination verschiedener Suchanfragen mittels Operatoren [8]. Lucene unterstützt sowohl einschließende als auch ausschließende Operatoren.

- Der Infix-Operator `AND` verknüpft zwei Suchklauseln in einer Konjunktion. Es werden alle Dokumente gefunden, die beide Suchklauseln enthalten. Statt `AND` kann auch die Zeichenfolge `&&` verwendet werden.

- Der Infix-Operator `OR` verknüpft zwei Suchklauseln in einer Disjunktion. Es werden alle Dokumente gefunden, die eine der beiden Suchklauseln enthalten. Statt `OR` kann auch die Zeichenfolge `||` verwendet werden.

- Der Prefix-Operator `+` bestimmt, dass alle Ergebnisse die entsprechende Suchklausel enthalten müssen.
- Der Prefix-Operator `NOT` bestimmt, dass alle Ergebnisse die entsprechende Suchklausel nicht enthalten dürfen. Anstatt `NOT` können auch die Zeichen `-` oder `!` verwendet werden. Der Operator `NOT` kann nicht alleine verwendet werden. Ein Abfrage muss immer mindestens eine positiv formulierte Suchklausel enthalten. Die Abfrage `NOT term1` liefert keine Ergebnisse.

Bei der Suche nach Begriffen mit Leerzeichen ist zu beachten, dass Anführungszeichen nicht funktionieren. Die Suche nach `feld:"term1 term2"` sucht im Feld `feld` nach `term1` und zusätzlich in allen Feldern nach `term1`. Die korrekte Suchanfrage wäre `feld:term1 feld:term2`.

Alle Sonderzeichen, die eine Funktion in der Syntax der Lucene-Abfragesprache haben, müssen mit dem Zeichen `\` maskiert werden, wenn sie außerhalb ihrer Funktion benutzt werden. Dazu gehören folgende Zeichen: `+ - && || ! () [] {} ^ " ~ * ? : \`

3.3.4 *Von der Suchanfrage zur Ergebnisliste*

Für die Abfrage eines Volltextindex, der zuvor aus einer Topic Map extrahiert wurde, steht die Klasse `TopicMapSearcher` zur Verfügung. Bei der Initialisierung der Klasse wird zunächst festgelegt, ob sich der zu durchsuchende Index im Hauptspeicher oder auf der Festplatte befindet. Zusätzlich kann die Klasse umfänglich konfiguriert werden (vgl. Anhang C).

Die Suche wird durch den Aufruf der Methode `search()` gestartet. Diese Methode erwartet die Suchanfrage als Parameter. Zusätzlich kann auch das ursprünglich indexierte Topic-Map-Objekt an die Methode übergeben werden. Ob dies geschieht, hat Auswirkungen auf den Typ der Ergebnisobjekte.

Innerhalb der Methode `search()` wird die Suchanfrage analysiert, wobei der Analyseprozess identisch zu dem Prozess ist, der beim Indexieren durchgeführt wird (vgl. Abschnitt 3.1.4.4). Nach der Analyse werden die Suchwörter in allen oder, falls angegeben, in einigen Feldern gesucht und die im Index gefundenen Dokumente als eine Liste von Suchergebnissen zurückgegeben.

Abhängig davon, ob beim Aufruf der Methode `search()` das passende Topic-Map-Objekt übergeben wurde, besteht die Ergebnisliste aus Objekten verschiedenen Typs. Wird das Topic-Map-Objekt nicht übergeben, so besteht die Ergebnisliste aus Objekten des Typs `ScoredResult`, welches nur grundlegende Informationen über das gefundene Topic enthält. Im anderen Fall besteht die Ergebnisliste aus Objekten des Typs `ScoredTopic`, welche zusätzlich das gefundene Topic als TMAPI-kompatibles

INFORMATION	SCORED-RESULT	SCORED-TOPIC
Das Dokument, welches im Index gefunden wurde	✓	✓
Das identifizierende Element des Topics aus dem Feld `si` oder dem Feld `sl` des gefundenen Dokuments	✓	✓
Die Namen des Topics aus den `name`-Feldern des gefundenen Dokuments	✓	✓
Das berechnete Gewicht des Dokuments	✓	✓
Eine Information darüber, ob das Topic ein Topic-Typ ist oder nicht.	✓	✓
Das Topic-Objekt, welches anhand des identifizierenden Elementes aus der Topic Map extrahiert wurde		✓

Tabelle 5: Die Ergebnistypen und ihre Informationen

Topic-Objekt enthalten. Dadurch wird eine flexiblere Weiterverarbeitung ermöglicht [35]. Die in den beiden Ergebnistypen enthaltenen Informationen werden in Tabelle 5 dargestellt.

3.3.5 *Gewichtung von Suchergebnissen*

Wie bereits in Abschnitt 1.1 deutlich wurde, soll eine Suchmaschine dem Nutzer das manuelle Durchsuchen von unübersichlich großen Datenmengen ersparen. Um dieses Ziel zu erreichen präsentiert die Suchmaschine eine Anzahl von Suchergebnissen, die anhand der Benutzereingabe aus dem Index ausgewählt wurden. Dabei entsteht das Problem, dass wiederum die Anzahl der Suchergebnisse unüberschaubar groß sein kann.

Um die Relevanz jedes einzelnen Suchergebnisses zu bestimmen, müsste der Nutzer jedes gefundene Dokument einzeln lesen. Diese mühselige Aufgabe kann die Suchmaschine dem Nutzer erleichtern, indem Suchergebnisse a priori bewertet und gewichtet werden, sodass die Einträge der Ergebnisliste nach ihrer berechneten Relevanz sortiert werden können.

Lucene berechnet das Gewicht g für jedes gefundene Dokument d, das zu einem Term t einer Anfrage q passt, auf folgende Weise:

$$g = \sum_{t \text{ in } q} (tf(t \text{ in } d) \times idf(t)^2 \times boost(f(t) \text{ in } d) \times lengthNorm(f(t) \text{ in } d)) \times coord(q, d) \tag{3.1}$$

Die einzelnen Bestandteile der Formel sind folgende:

- Die Termfrequenz `tf(t in d)` repräsentiert die Häufigkeit, mit der ein Term `t` im Dokument `d` auftritt [1].
- Die inverse Dokumentenhäufigkeit `idf(t)` gibt an, ob ein Term `t` ein Dokument gut beschreibt. Das ist der Fall, wenn der Term `t` nur in wenigen Dokumenten des Index auftritt [1].
- Das Basisgewicht `boost(f(t) in d)` eines Feldes `f` des Dokumentes `d`, in dem der Term `t` auftritt. Das Basisgewicht wird während des Indexierens festgelegt (vgl. Abschnitt 3.2.4).
- Der Normalisierungswert `lengthNorm(f(t) in d))` wird während des Indexierens berechnet und gibt die Anzahl der Terme in einem Feld `f` eines Dokuments `d` an [27].
- Der Koordinierungsfaktor `coord(q, d)` gibt die Anzahl der Terme aus der Anfrage `q` wieder, die im Dokument `d` enthalten sind [27].

Aus diesen Teilgewichten wird für jedes Suchergebnis das Gesamtgewicht ermittelt.

3.3.6 *Suche in virtuell zusammengeführten Topic Maps*

Für die Suche in virtuell zusammengeführten Topic Maps stehen verschiedene Methoden zur Verfügung (vgl. Abschnitt 3.2.6). Liegt ein einzelner Index vor, der aus einer vorhergehenden Zusammenführung stammt, so kann dieser wie jeder andere monolithische Index durchsucht werden.

Liegen jedoch für die virtuell zusammengeführten Topic Maps jeweils einzelne Indexe vor, so müssen diese einzeln durchsucht werden. Dies kann mit einem mehrfachen Aufruf der Methode `search()` erfolgen. Wird dabei jeweils das virtuell zusammengeführte Topic-Map-Objekt übergeben, so enthält die Ergebnisliste korrekt zusammengeführte Ergebnisse, da eventuell auftretende Dubletten bereits bei der Extraktion der Ergebnis-Topics aus der virtuell zusammengeführten Topic Map entfernt werden.

Wird jedoch kein Topic-Map-Objekt übergeben, so kann die Ergebnismenge Dubletten beinhalten. Die Filterung und Zusammenführung dieser mehrfachen Suchergebnisse obliegt dann der Applikation, in welche die Topic-Map-Suchmaschine eingebettet wurde.

Falls nicht für alle virtuell zusammengeführten Topic Maps ein Index vorliegt, werden lediglich die vorliegenden Indexe durchsucht. Nachdem die Suche durchgeführt wurde, kann über einen Aufruf der Methode `getMtmSearchStatus()` eine Statusinformation abgerufen werden, die eine der folgenden Bedeutungen hat:

- Der Status `INDEX_COMPLETE` gibt an, dass für alle virtuell zusammengeführten Topic Maps ein Index vorhanden war.
- Der Status `INDEX_INCOMPLETE` bedeutet, dass nicht für alle virtuell zusammengeführten Topic Maps ein Index vorhanden war.
- Der Wert `INDEX_EMPTY` gibt an, dass für keine der virtuell zusammengeführten Topic Maps ein Index vorhanden war.

Die Methode, mit der die Indexe der virtuell zusammengeführten Topic Map durchsucht werden, hat keinen Einfluss auf die Berechnung der Gewichte der Ergebnisse.

3.3.7 *Statistische Informationen*

Die Methode `getStatistics()` der Klasse `TopicMapsSearcher` liefert einige statistische Werte über einzelne Indexe, die sowohl bei der Fehlersuche als auch beim Vergleich verschiedener Indexe nützlich sein können. Die einzelnen Werte sind folgende:

- Der Wert `diskSpace` enthält den vom Index verbrauchten Speicher oder Festplattenplatz in Byte.
- Die Anzahl der im Index enthaltenen Dokumente wird mit dem Wert `numDocs` repräsentiert. Da während des Indexierens für jedes Topic genau ein Dokument angelegt wird, muss dieser Wert mit der Anzahl der Topics in der indexierten Topic Maps übereinstimmen.
- Während des Indexierens wird der Wert `indexTime` aufgezeichnet, welcher die beim Indexieren verbrauchte Zeit in Millisekunden darstellt.

4

FACETTIERTE SUCHE IN TOPIC MAPS

Die klassische Volltextsuche, wie sie in Kapitel 3 beschrieben wird, adressiert das Paradigma der *informationellen Suche*. Bei dieser Form der Informationsgewinnung ist der Benutzer in der Datenbasis auf der Suche nach Elementen, die er nicht kennt, von denen er jedoch annehmen kann, dass sie mit der Eingabe der richtigen Suchbegriffe im vorliegenden Volltextindex gefunden werden können. Das Paradigma der *navigierenden Suche* wird jedoch durch die klassische Volltextsuche nicht abgedeckt. Bei der navigierenden Suche werden die Elemente einer zuvor erhaltenen Menge von Suchergebnissen als Ausgangspunkte für eine weitergehende Erforschung der Datenbasis genutzt [38].

Dieses Suchparadigma kann mit Hilfe von Facetten behandelt werden. Als Facetten werden „klar definierte, sich gegenseitig ausschließende[1] und in ihrer Gesamtheit vollständig beschreibende Aspekte, Eigenschaften und Charakteristika“ einer bestimmten Informationsressource bezeichnet [33, 36]. Beispielsweise besitzt eine Person eine Namensfacette, eine Geburtstagsfacette und eine Wohnortfacette, die jeweils klar definiert sind und völlig voneinander unabhängige Charakteristika einer Person beschreiben, wodurch diese drei Facetten zur eindeutigen Identifizierung einer Person genutzt werden können.

Zusätzlich zur Problematik der navigierenden Suche kann mit dem Einsatz von Facetten ein weiteres Problem adressiert werden. Um eine möglichst hohe Trefferquote zu erhalten, geben viele Suchmaschinenbenutzer intuitiv eher allgemeine Begriffe in die Suchmaske ein. Dann wird zwar keine relevante Informationsressource bei der Volltextsuche übersehen, allerdings wird der Benutzer mit einer unüberschaubar großen Zahl an Suchergebnissen konfrontiert. Facetten bieten dann die Möglichkeit, die Menge der Suchergebnisse anhand vorher bestimmter Eigenschaften der Informationsressourcen einzuschränken.

4.1 DER FACETTENBEGRIFF IM TMDM

Das Konzept der Facetten bei Topic Maps wurde von Steve Pepper bereits im Jahr 2000 erörtert [29]. Die eingeführten Facetten hatten jedoch eine andere Aufgabe. Sie sollten nicht adressierbaren Topics beschreibende Metadaten zuordnen. Daraus ergab sich

1 Dass sie sich „gegenseitig ausschließen“ bedeutet, dass zwei Facetten komplett voneinander unabhängige Eigenschaften einer Informationsressource beschreiben.

die Möglichkeit, Topics nach diesen Facetten zu filtern, ähnlich wie das auch in diesem Kapitel beschrieben wird.

Der im Jahr 2001 veröffentlichte Standard XTM 1.0 schreibt, wie alle späteren Weiterentwicklungen auch, für nicht adressierbare Topics allerdings die Verwendung einer Gegenstandsbeschreibung vor [37]. Damit wurde das Konzept der Beschreibung eines nicht adressierbaren Topics durch Metadaten überflüssig, sodass die ursprünglich von Steve Pepper eingeführten Facetten heute nicht mehr Teil der Topic-Maps-Spezifikation sind.

4.2 QUALITÄTSMETRIKEN FÜR FACETTEN

Um bei ihrer Aufgabe in der navigierenden Suche effizient genutzt werden zu können, müssen Facetten gewisse Kriterien erfüllen. Die *Navigationsqualität* einer Facette hängt im Wesentlichen von drei messbaren Eigenschaften ab: Balance des Navigationsbaumes, sinnvoll beschränkte Kardinalität sowie Mindestfrequenz in ihrem Auftreten [28].

Alle Metriken werden auf das Intervall $[0 \dots 1]$ normalisiert und können anschließend durch eine gewichtete Multiplikation zu einer endgültigen Bewertung zusammengefasst werden. Niedrig bewertete Facetten sollten jedoch nicht komplett verworfen werden, da sie für den Benutzer intuitiv erscheinen können, selbst wenn sie objektiv komplett ineffizient sind.

4.2.1 *Balance einer Facette*

Der Navigationsbaum einer Facette setzt sich aus der Menge der Navigationsmöglichkeiten, also den verschiedenen auftretenden Werten dieser Facette, zusammen. Dabei ist ein Navigationsbaum umso effizienter, je besser seine Äste ausbalanciert sind. Die Balance des Navigationsbaumes einer Facette $\mathtt{balance}(f)$ wird berechnet aus der Verteilung $n_i(v_j)$ der Werte über die zugehörigen Instanzen als mittlere Abweichung vom Vektormittel μ und wird mit der Worst-Case-Abweichung auf $[0 \dots 1]$ normalisiert, wobei v die Anzahl verschiedener Werte darstellt:

$$\mathtt{balance}(f) = \frac{\sum_{j=1}^{v} |n_i(v_j) - \mu|}{(v-1)\mu + (I-\mu)} \tag{4.1}$$

4.2.2 *Kardinalität einer Facette*

Eine zur Navigation geeignete Facette hat eine beschränkte Menge an Werten, aus denen der Nutzer zur Navigation auswählen kann. Diese Menge liegt im Allgemeinen zwischen zwei und zehn, da mit steigender Anzahl an Auswahlmöglichkeiten die Auswahl unübersichtlicher wird. Die Kardinalität einer Facette

$\mathtt{card}(f)$ wird aus der Anzahl der verschiedenen Objekte $n_0(f)$ für eine gegebene Facette f berechnet und mit einer Normalverteilung normalisiert, die von den Parametern η und ρ abhängig ist:

$$\mathtt{card}(f) = \begin{cases} 0 & \text{wenn } b_0(f) \leqslant 1 \\ \exp^{-\frac{(n_0(f)-\eta)^2}{2\rho^2}} & \text{sonst} \end{cases} \tag{4.2}$$

Der Parameter η kann dabei abhängig von $n_0(f)$ gewählt werden, etwa $0.1 \times n_0(f)$, wogegen der Parameter ρ zwischen $1,0$ und $2,5$ liegt.

4.2.3 *Frequenz einer Facette*

Damit Facetten effektiv zur Navigation genutzt werden können, müssen sie mehrfach auftreten. Je mehr Topics abgedeckt werden, umso besser teilt die entsprechende Facette den Informationsraum. Die Frequenz einer Facette $\mathtt{freq}(f)$ wird als Anzahl der Instanzen $n_i(f) = |\mathtt{exists}(f)|$ einer Topic Map berechnet, für die eine Facette f definiert wurde. Sie wird mit der Gesamtanzahl der Instanzen n_i normalisiert:

$$\mathtt{freq}(f) = \frac{n_i(f)}{n_i} \tag{4.3}$$

Diese drei Metriken können bereits bei der Extraktion der Facetten aus der Topic Map vorberechnet werden. Dafür steht die Klasse `FacetEvaluator` zur Verfügung, welche in der Methode `addFacet()` die in diesem Abschnitt eingeführten Metriken implementiert.

Die eingeführten Metriken Balance, Kardinalität und Frequenz sind objektiv messbare Kriterien für alle Arten von Facetten. Sie nehmen allerdings keinen Bezug auf das eigentliche *Informationsbedürfnis*[2] der Nutzer und haben deshalb keinerlei Aussagekraft bezüglich der subjektiven Zufriedenheit der Nutzer.

4.3 FACETTEN UND TOPIC MAPS

Ohne Beschränkung der Allgemeinheit kann die Menge der Topics in einer gut modellierten Topic Map in zwei Untermengen aufgeteilt werden:

- Die Menge aller Topic-Typen T enthält nur Topics mit Ontologieinformationen, also Topics, die über Supertyp-Subtyp- oder Typ-Instanz-Beziehungen miteinander verbunden sind.

2 Das Informationsbedürfnis beschreibt die einzelne Information, die ein Nutzer innerhalb der Datenbasis zu finden hofft.

- Die Menge aller Topicinstanzen I enthält alle Topics, in denen Instanzdaten gespeichert werden.

Diese beiden Mengen sind nicht unbedingt disjunkt. Allerdings ist die Unterscheidung notwendig, da die Definition domänenspezifischer Facetten nur für Elemente der Menge T möglich ist, während die Facetten selbst für Topics aus der Menge I generiert werden. Eine Facette f eines Topics $i \in I$ ist definiert als das Tripel

$$f = (\mathrm{id}(i), n, v \in V) \quad , \tag{4.4}$$

wobei $\mathrm{id}(i)$ einen eindeutigen Bezeichner des Instanztopics i, n den Namen und v den Wert der Facette aus der Menge aller Facettenwerte V darstellt.

4.3.1 *Extraktion generischer Facetten*

Nach dem Topic-Maps-Paradigma wird jeder Aussagegegenstand durch ein Topic repräsentiert. Dieses Topic trägt auch die Eigenschaften des Aussagegegenstandes, wie Benennungen, Belegstellen oder Beziehungen zu anderen Aussagegegenständen. Alle diese Konstrukte können in einem generischen Prozess in Facetten überführt werden, wobei insbesondere Belegstellen und Beziehungen gut für die Generierung von Facetten geeignet sind [39]. Dies gilt allerdings nur für Belegstellen mit kurzem Inhalt. Größere Texte sind nicht als Werte für Facetten geeignet. Neben Belegstellen und Beziehungen können allerdings auch Benennun-

KONSTRUKT	NAME n	WERT v
Topic-Typ	`"topic_type"`	Benennung des Topic-Typs
Benennung	Benennung des Namenstyps	Wert der Benennung selbst
Belegstelle	Benennung des Typs der Belegstelle	Wert der Belegstelle selbst
n-stellige Beziehung ($n = 1$)	Name des Typs der gespielten Rolle	`"true"`
n-stellige Beziehung ($n > 1$)	Name des Typs der gespielten Rollen der anderen Teilnehmer der Beziehung	Name des rollenspielenden Topics

Tabelle 6: Übersicht der für die Facettierung eines Topics genutzten Topic-Map-Konstrukte

gen und Topic-Typen für die Erstellung von Facetten genutzt werden [7, 28].

Zur Erstellung generischer Facetten für die Topics einer Topic Map steht die Klasse `TopicMapFacetter` zur Verfügung. Diese ist von der Klasse `TopicMapIndexer` abgeleitet und speichert die generierten Facetten als Einträge eines Lucene-Volltextindex ab. Die Erstellung wird durch den Aufruf der Methode `index()` gestartet, wobei zunächst die Facetten aller Topics der als Parameter übergebenen Topic Map erstellt werden.

Die Extraktion der Facetten eines Topics wird in der Methode `createTopicFacets()` der Klasse `FacetFactory` durchgeführt, welche das Topic als Parameter übernimmt. Innerhalb dieser Methode werden die Facetten des Topics aus den Topic-Typen, den Benennungen und Belegstellen des Topics sowie den Rollen extrahiert, welche dieses Topic spielt. Eine genaue Übersicht über die extrahierten Werte bietet Tabelle 6.

Nachdem der Extraktionsprozess abgeschlossen ist, werden die Facetten anhand der in Abschnitt 4.2 eingeführten Metriken evaluiert und gemeinsam mit den Ergebnissen der Evaluation in einen Lucene-Index gespeichert. Dabei werden die folgenden Felder verwendet:

- Um das Topic identifizieren zu können, zu dem die Facette gehört, wird ein identifizierendes Element des Topics indexiert. Der Feldname lautet dabei `si` für indirekte Gegenstandsanzeiger, `sl` für direkte Gegenstandsanzeiger und `ii` für Identifizierungen.
- Der Name der Facette wird im Feld `klass` hinterlegt. Um die spätere Identifizierung des Namens zu erleichtern, wird außerdem ein identifizierendes Element des Namens im Feld `klass_identifier` gespeichert. Der Typ dieses Elements wird im Feld `klass_identifier_type` hinterlegt.
- Der Facettenwert findet sich im Feld `value`. Das entsprechende identifizierende Element wird im Feld `value_identifier` gespeichert. Der Typ dieses Elements wird im Feld `value_identifier_type` abgelegt.
- Das Feld `weight` enthält das vorberechnete Gewicht der Facette.
- Darüber hinaus gibt es beliebig viele `scope`-Felder, die identifizierende Elemente enthalten, die den Gültigkeitsbereich des Namens bestimmen.

4.3.2 *Definition domänenspezifischer Facetten*

Generische Facetten eignen sich besonders gut für Topic-Maps-Ontologien mit grobgranularen Strukturen, in denen alle Eigen-

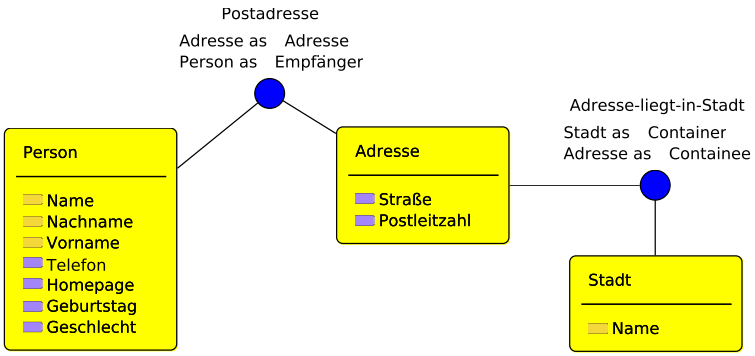

Abbildung 7: Beispielmodell mit drei Topic-Typen: Person, Adresse und Stadt. Um eine Wohnort-Facette für eine Person zu modellieren muss die indirekte Beziehung zwischen einer Person und der dazugehörigen Stadt traversiert werden.

schaften der Topics durch Belegstellen oder *direkte Assoziationen* repräsentiert werden. Direkte Assoziationen sind n-stellige Beziehungen, deren Rollenspieler direkt und ohne Zwischenstation miteinander verbunden sind (beispielsweise die Beziehungen zwischen Personen und Adressen in Abbildung 7).

Weniger gut eignen sich generische Facetten für feingranulare Ontologien, in denen nicht alle Eigenschaften eines Topics durch direkte Beziehungen erreichbar sind. Beispielsweise kann der Wohnort einer Person in Abbildung 7 nur bestimmt werden, indem die *indirekte Beziehung* zwischen einer Person und der dazugehörigen Stadt über die Zwischenstation des Adresstopics traversiert wird. Im Allgemeinen können indirekte Beziehungen auch als eine Folge direkter Beziehungen angesehen werden. Dadurch würde die explorative Navigation des Nutzers allerdings in viele kleine Schritte zerfallen und der Nutzer würde zu viel Aufwand benötigen, um das Gesuchte zu finden [38].

Für die Definition domänenspezifischer Facetten wird ein Ausdruck benötigt, der sowohl die effektive Vorberechnung als auch die Zuweisung der vorberechneten Werte an die Instanzen eines gegebenen Topic-Typs ermöglicht. Dies ermöglicht die Topic-Maps-Abfragesprache TMQL. Allerdings sind die verfügbaren Sprachkonstrukte der TMQL viel umfangreicher, als für die Definition von domänenspezifischen Facetten notwendig ist. Von den in Abschnitt 2.3 eingeführten TMQL-Sprachdialekten werden lediglich die Pfad-Ausdrücke benötigt.

Ein Pfad-Ausdruck repräsentiert eine Abfolge von Navigationsschritten durch den abstrakten bidirektionalen Graphen einer Topic Map. Ausgehend von einem eindeutig identifizierbaren Konstrukt der Topic Map, können mit Navigationsschritten entlang vordefinierter Achsen einzelne Elemente aus der Topic Map extrahiert werden [18]. Diese Elemente können dann als Werte für die generierten Facetten dienen.

Entsprechend dem Beispiel aus Abbildung 7 kann eine Wohnort-Facette mit dem folgenden Pfad-Ausdruck definiert werden:

```
http://psi.example.com/person
  >> traverse http://psi.example.com/mailing-address
  >> traverse http://psi.example.com/container-containee-address
  >> characteristics tm:name
```

Wird dieser Ausdruck auf die Menge der Instanzen I_{Person} des Topic-Typs t_{Person} angewandt, so enthält das Ergebnis den Namen der Stadt, in der die jeweilige Person lebt. Dies wird erreicht, indem der gegebene indirekte Gegenstandsanzeiger von t_{Person} durch den indirekten Gegenstandsanzeiger des jeweiligen Instanztopics ersetzt wird.

Domänenspezifische Facetten können mit der Methode `createTMQLFacets()` der Klasse `TopicMapFacetter` erzeugt werden. Diese Methode erwartet die zu verarbeitende Topic Map, den Namen der zu erstellenden Facetten sowie den notwendigen TMQL-Pfadausdruck als Parameter. Zunächst wird der TMQL-Pfadausdruck auf Ausführbarkeit geprüft. Fällt die Prüfung positiv aus, so wird die Pfadabfrage für alle ermittelten Instanzen des gegebenen Topic-Typs ausgeführt.

Der mit der TMQL-Abfrage ermittelte Wert kann dann als Wert v einer neuen Facette verwendet werden. Da der Inhalt dieses Wertes eine Zeichenkette ist, muss das Ergebnis der TMQL-Abfrage eine Belegstelle, eine Benennung oder eine sonstige Zeichenkette darstellen. Belegstellen und Benennungen werden mit der TMAPI-Methode `getValue()` automatisch in eine Zeichenkette überführt.

Nachdem alle Facetten erstellt wurden, können diese über die Methode `update()` der Klasse `TopicMapFacetter` dem Facettenindex hinzugefügt werden. Nach dem Update kann die Methode `reevaluate()` aufgerufen werden. Diese Methode führt eine komplette Reevaluation des Facettenindex durch. Dies ist notwendig, da sich durch das Hinzufügen neuer Facetten die Gewichte der bereits vorhandenen Facetten ändern können.

4.3.3 *Suche im Facettenindex*

Bei der Abfrage des Facettenindex können zwei unterschiedliche Abfragemodi benutzt werden: *facettierte Klassifikation* und *facettierte Suche* [30]. Durch die Speicherung der Facetten in einem Volltextindex besteht einerseits die Möglichkeit einer facettierten Suche. Dabei werden die Facetten mit der in Abschnitt 3.3.4 vorgestellten Volltextsuche abgefragt. Dazu wird die Klasse `TopicMapSearcher` so konfiguriert, dass die Methode `search()` direkt auf den Facettenindex zugreift.

Zum anderen kann der Facettenindex darüber hinaus für eine facettierte Klassifikation genutzt werden, um die Facetten

einer Menge von Suchergebnissen des Volltextindex zu bestimmen. Dazu dient die Methode `searchWithFacets()` der Klasse `TopicMapSearcher`. Diese Methode erwartet die Suchanfrage als Parameter und führt in einem ersten Schritt eine Suche auf dem Volltextindex durch (vgl. Abschnitt 3.3.4).

Für jedes der aus dieser Suche resultierenden Ergebnisse wird anschließend eine Suche auf dem Facettenindex ausgeführt. Bei dieser Suche werden die in den Feldern `si`, `sl` und `ii` gespeicherten identifizierenden Elemente durchsucht, wobei alle Facetten abgerufen werden, deren identifizierendes Element mit dem identifizierenden Element des jeweiligen Suchergebnisses übereinstimmt.

Ist dieser Suchvorgang abgeschlossen, werden die gefundenen Facetten zusammen mit den ursprünglichen Suchergebnissen in ein Ergebnisobjekt des Typs `FacettedResult` umgewandelt und zurückgegeben.

4.3.4 *Facetten aktualisieren*

Um die Indexqualität des Facettenindex zu gewährleisten, können einzelne Facetten reindexiert bzw. inkrementell zum Facettenindex hinzugefügt werden. Dazu bietet die Klasse `TopicMapFacetter` die Methode `update()`, welche die neuen Facetten, das Herkunftstopic der Facetten sowie einen Update-Typ (vgl. Tabelle 4) als Parameter erwartet. Da die Gewichte aller Facetten von den Gewichten aller anderen Facetten abhängen, muss nach einem Update die Methode `reevaluate()` aufgerufen werden.

5

SUCHE IN TOPIC-MAPS-GETRIEBENEN WEBPORTALEN

Webportale, die auf der Basis von Topic Maps arbeiten, sind häufig nur mit einer klassischen Volltextsuche ohne semantische Erweiterungen ausgestattet. Viele Möglichkeiten, die sich durch die Strukturiertheit der Daten und die zugrunde liegende semantische Technologie für die effiziente Bearbeitung von Suchanfragen ergeben, bleiben dadurch ungenutzt. Durch eine konsequente Ausnutzung der semantischen Datenstruktur können jedoch in allen wesentlichen Aufgabenbereichen von Suchmaschinen wesentliche Verbesserungen erzielt werden [22]:

- Bereits bei der Erstellung des Index gehen viele semantische Informationen verloren, da diese in einem klassischen Volltextindex nicht dargestellt werden können. Dieses Problem kann durch die Nutzung eines speziellen Topic-Maps-orientierten Index gelöst werden, wie er in Kapitel 3 vorgestellt wird.

- Für klassische Volltextsuchmaschinen hat sich ein allgemein anerkannter Standard für die Präsentation von Suchergebnissen gebildet[1]. Dieser De-facto-Standard ist zwar für schwach strukturierte Texte geeignet, ignoriert jedoch die Möglichkeiten, die stark strukturierte Datenquellen wie Topic Maps bieten. Eine Präsentation mittels Facetten, in denen sich die zugrunde liegende Ontologie widerspiegelt, eröffnet dagegen dem Benutzer die Möglichkeit des explorativen Durchsuchens der Suchergebnisse (vgl. Kapitel 4).

Um diese Probleme zu adressieren, wird eine Topic-Map-basierte Suchmaschine benötigt, die sich in die Umgebung moderner Webapplikationen einbetten lässt und dabei das volle Potential von Topic Maps ausschöpft.

Eine solche Umgebung moderner Webapplikationen bildet das Webapplikations-Framework *Ruby on Rails*, das auch die Grundlage für den Topic-Maps-Browser *Maiana* bildet. In ihn wurde die in den Kapiteln 3 und 4 entwickelte semantische Topic-Maps-Suchmaschine mit Hilfe der Middleware *JRuby Topic Maps (JRTM)* prototypisch integriert.

1 Dieser umfasst meist den Seitentitel mit Link und einen kurzen Anriss des Textes mit den gefundenen Suchwörtern (vgl. Abbildung 1).

5.1 JRUBY TOPIC MAPS

Das Topic Maps Application Programming Interface (TMAPI) bildet die Grundlage vieler Topic-Maps-verarbeitender Systeme. Ursprünglich nur für Java entwickelt, existieren heute auch TMAPI-Implementierungen in anderen Programmiersprachen, wie etwa PHP. Diese nutzen jedoch nicht den ursprünglichen Java-Code. Im Gegensatz dazu erweitert *JRuby Topic Maps (JRTM)* die ursprüngliche TMAPI-Implementierung in Java, um ein abstraktes API für Topic Maps in Ruby[2] zu beschreiben [3]. Dies wird durch die erweiterten Möglichkeiten von JRuby ermöglicht. Im Gegensatz zur ursprünglichen C-basierten Implementierung von Ruby läuft JRuby in einer *JVM*[3] und ist damit in der Lage, externe Java-Bibliotheken zu laden. Die geladenen Klassen und Methoden können dann direkt aus dem Ruby-Code heraus aufgerufen werden.

Die Architektur von JRTM besteht aus mehreren Paketen, die im Ruby-Umfeld als RubyGems bezeichnet werden (vgl. Abbildung 8). Das zentrale Paket ist `rtm-javatmapi`. Dieses Paket stellt die Schnittstelle zwischen den darauf aufsetzenden JRuby-Applikationen und dem TMAPI dar. Die Basis bildet dabei eine Topic-Maps-Engine, welche das TMAPI implementiert. JRTM kann mit allen gängigen Topic-Map-Engines arbeiten[4].

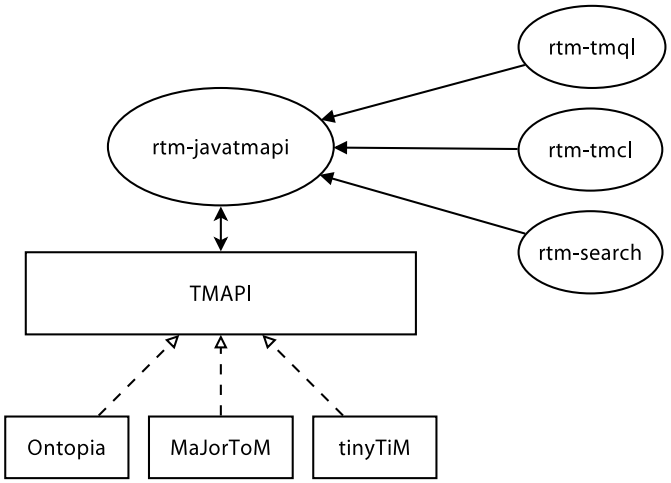

Abbildung 8: Die verschiedenen Komponenten von JRTM. Die Topic-Map-Engines – hier beispielhaft Ontopia, MaJorToM und tinyTiM – implementieren das Topic Maps Application Programming Interface (TMAPI). JRuby-Pakete, wie *rtm-tmql*, *rtm-tmcl* oder *rtm-search*, bieten erweiterte Funktionalität.

2 Ruby ist eine moderne höhere Programmiersprache, die Mitte der 1990er-Jahre von Yukihiro Matsumoto entworfen wurde.

3 Java Virtual Machine.

4 Zum Zeitpunkt der Erstellung dieser Arbeit existierten Module für CouchTM, Hatana, MaJorToM, Ontopia, SesameTM, tinyTiM und TM2JDBC.

Neben dem Paket `rtm-javatmapi` existieren eine Reihe weiterer Pakete, die als Mixins[5] fungieren und zusätzliche Funktionalität bereitstellen. Beispiele dafür sind `rtm-tmql` und `rtm-tmcl`. Diese Pakete stellen die Funktionalitäten der Topic-Maps-Abfragesprache TMQL bzw. der Topic-Maps-Schemasprache TMCL für JRuby zur Verfügung. Daneben gibt es die Pakete `rtm-search` und `rtm-search_hatana`, welche die in dieser Arbeit vorgestellten Index-, Such- und Facettierungsmethoden für JRuby zur Verfügung stellen:

- Im Paket `rtm-search` befinden sich Methoden, die das Indexieren, die Suche und das Facettieren von TMAPI-kompatiblen Topic Maps ermöglichen.
- Das Paket `rtm-search_hatana` enthält spezielle Methoden, die für die Suche in Hatana-Topic-Maps nötig sind. *Hatana* ist eine Bibliothek, die Funktionen für das virtuelle Zusammenführen von Topic Maps anbietet.

Diese Unterteilung ist notwendig, da die semantische Topic-Maps-Suchmaschine nach ihrer Fertigstellung als Open-Source-Software veröffentlicht werden soll. Die Bibliothek *Hatana* kann jedoch nicht Teil dieser Veröffentlichung sein, da sie nicht unter einer Open-Source-Lizenz steht. Daher wurden die Hatana-spezifischen Funktionen extrahiert, sodass der nicht mit Hatana arbeitende Teil dennoch veröffentlicht werden kann.

5.1.1 *RTM-Search*

Wie bei Ruby-Bibliotheken üblich, werden alle Klassen in Modulen zusammengefasst. Im Paket `rtm-search` werden zwei Klassen definiert:

- Die Klasse `RTM::Search::Indexer` stellt zunächst einige Methoden bereit, mit denen die in Anhang C beschriebenen Konfigurationen der Klasse `TopicMapIndexer` bearbeitet werden können. Darüber hinaus werden die folgenden Methoden definiert:
  - `index` – Führt die Indexierung einer gegebenen Topic Map durch (vgl. Abschnitt 3.2.4).
  - `index_directory` – Findet das Index-Verzeichnis zu einer gegebenen Basisadresse.
  - `index_facets` – Erstellt den Facettenindex für eine gegebene Topic Map (vgl. Abschnitt 4.3.1).

5 Als Mixin wird in der objektorientierten Programmierung ein mehrfach verwendbares Bündel von Funktionen bezeichnet, das zu einer Klasse hinzugefügt werden kann.

  - `update` – Führt für ein gegebenes Topic ein inkrementelles Update des Index durch (vgl. Abschnitt 3.2.5).
  - `destroy_index` – Löscht den zur gegebenen Basisadresse gehörenden Index (vgl. Abschnitt 3.2.7).

- In der Klasse `RTM::Search::Searcher` werden Methoden für die Konfiguration der Klasse `TopicMapSearcher` bereitgestellt. Zusätzlich existieren folgende Methoden:
  - `search` – Führt eine Suche mit der gegebenen Anfrage auf dem Index aus, der durch die gegebene Basisadresse identifiziert wird (vgl. Abschnitt 3.3.4).
  - `search_facets` – Führt eine Suche auf dem Facettenindex aus, der zu dem Index gehört, der durch die gegebene Basisadresse identifiziert wird (vgl. Abschnitt 4.3.3).
  - `statistics` – Liefert einige statistische Angaben zu dem Index mit der gegebenen Basisadresse (vgl. Abschnitt 3.3.7).

Die oben angeführten Methoden werden darüber hinaus auch für die Klassen `RTM::TopicMap` und `RTM::Topic` definiert. Damit können diese Methoden auch direkt an Instanzen dieser Klassen aufgerufen werden. Dort wird dann zunächst eine Instanz der benötigten Klasse aus dem Modul `RTM::Search` initialisiert. Danach wird die entsprechende Methode dieser Instanz aufgerufen. Damit enthalten die Indexierungs- und Suchmethoden der Klassen `RTM::TopicMap` und `RTM::Topic` keine eigene Funktionalität, sondern dienen lediglich als eine komfortable Hülle.

5.1.2 *RTM-Search_Hatana*

Das Untermodul RTM-Search_Hatana definiert keine eigenen Methoden. Es dient lediglich dazu, die ebenfalls ausgegliederte Java-Bibliothek mit der Suchfunktionalität für virtuell zusammengeführte Topic Maps zu laden (vgl. Abschnitt 3.3.6). Dadurch werden die Java-Methoden dieser Bibliothek in JRTM sichtbar und können aus der JRuby-Umgebung heraus genutzt werden.

5.2 MAIANA

Maiana ist eine auf JRTM basierende Webapplikation, die dem Benutzer das Hochladen, Betrachten und Exportieren von Topic Maps ermöglicht [34]. In diesem Umfeld entsteht eine Vielzahl von Informationsbedürfnissen, die mit einer Volltextsuche befriedigt werden können. Beispielsweise interessieren sich Nutzer für die Inhalte der hochgeladenen Topic-Maps oder die Profile anderer Nutzer. Daher wurde das Paket `rtm-search` in Maiana in-

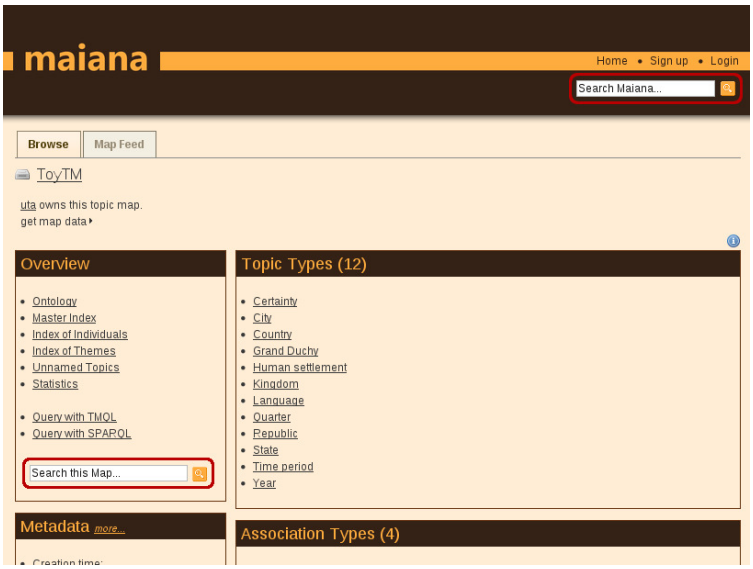

Abbildung 9: Hauptansicht einer Topic Map in Maiana. *Oben rechts* befindet sie die Seitensuche, *unten links* befindet sich die Topic-Map-Suche.

tegriert, wobei zwei verschiedene Varianten der Suche entstanden (vgl. Abbildung 9):

- Mit der *Seitensuche* können neben den Benutzernamen und den Namen aller für den Nutzer einsehbaren Topic Maps auch die für den Nutzer einsehbaren Topic Maps selbst durchsucht werden.
- Die *Topic-Map-Suche* dient zum Durchsuchen einer einzelnen Topic Map.

Neben JRTM basiert Maiana hauptsächlich auf Ruby on Rails, einem seit 2004 existierenden Webapplikations-Framework. Ruby on Rails folgt dabei der *Model-View-Controller-Architektur*. Diese Architektur trennt die Struktur einer Software-Architektur in die drei Einheiten Datenmodell (engl. model), Präsentation (engl. view) und Programmsteuerung (engl. controller) auf. Entsprechend dieses Modells wurde auch die Such-Funktionalität in Maiana in diese drei Einheiten getrennt. Das zentrale Datenmodell für die Suche ist im `Search`-Modell implementiert. Dieses Datenmodell bietet Zugriff auf die Indexierungs- und Such-Funktionen des Paketes `rtm-search` sowie Funktionen zur Fehleranalyse und zur Aufbereitung der Suchergebnisse. Anfragen für die Seitensuche werden durch den `SearchController` verarbeitet, die Verarbeitung der Anfragen an die Topic-Map-Suche wurde in den `EndpointsController`[6] integriert. Für die Anzeige der Suchergebnisse sind verschiedene Präsentationen verantwortlich.

6 Aufgrund der vielfältigen Varianten, mit denen Topic Maps in Maiana importiert werden können, wurde der allgemeine Oberbegriff *Endpoint* gewählt.

5.2.1 *Indexierung in Maiana*

Bevor die Indexe der Topic Maps in Maiana durchsucht werden können, müssen die entsprechenden Indexe erstellt werden. Dies geschieht, während des Ladens einer Topic Map[7]. Dazu wird eine neue Instanz des Rails-Modells `Search` initialisiert, die wiederum die Topic Map an eine neue Instanz der Klasse `RTM::Search::Indexer` übergibt.

5.2.2 *Die Seitensuche*

Um allen Benutzern eine schnelle Übersicht über die in Maiana gespeicherten Topic Maps zu geben, wurde die Seitensuche implementiert. Diese Suche greift auf alle Indexe der Topic Maps zu, die für den jeweiligen Benutzer sichtbar sind. Darüber hinaus werden auch die Namen der für den Nutzer sichtbaren Topic Maps sowie die Namen der Maiana-Benutzer selbst durchsucht.

Diese Funktionalität wurde in der Methode `fulltext_search` des `SearchControllers` implementiert. Sie nimmt die Suchanfrage des Benutzers entgegen. Danach wird die Basisadresse jeder durchsuchbaren und für den Nutzer sichtbaren Topic Map abgefragt. Für jede dieser Basisadressen wird dann eine Instanz des Rails-Modells `Search` erzeugt, welche die Suchanfrage an eine neue Instanz der Klasse `RTM::Search::Searcher` weiterleitet.

Darüber hinaus wird die Datenbank, welche die Namen der Benutzer und die Namen der Topic Maps enthält, ebenfalls durchsucht. Die drei Teilergebnisse werden anschließend für den Benutzer übersichtlich aufgelistet (vgl. Abbildung 10). An erster Stelle stehen dabei die Suchergebnisse aus den verschiedenen

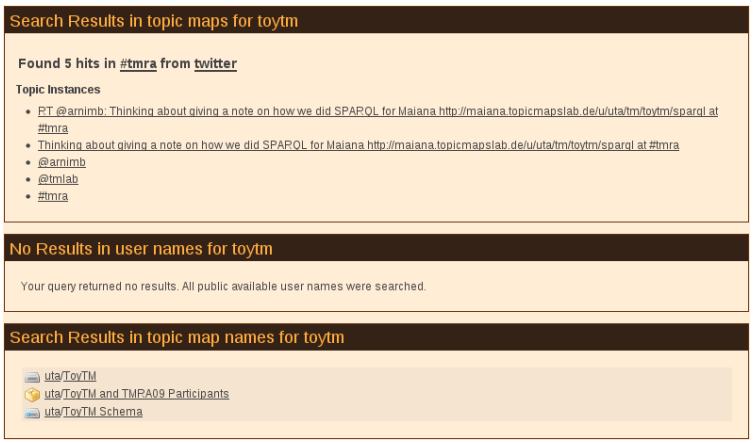

Abbildung 10: Ergebnisse der Seitensuche mit dem Suchbegriff *toytm*.

7 Zum Zeitpunkt der Erstellung dieser Arbeit wurden die einzelnen Topic Maps als Dateien auf der Festplatte verwaltet und beim ersten Zugriff in den Speicher geladen.

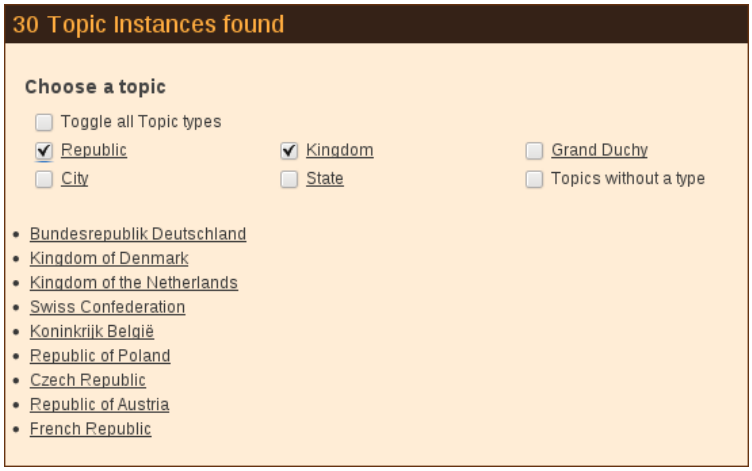

Abbildung 11: Ergebnisse der Topic-Map-Suche mit Topic-Typ-Facetten, mit Einschränkung auf Topics vom Typ „Republic“ und „Kingdom“

Topic Maps, geordnet nach Topic Map und kategorisiert in Topics mit und ohne Topic-Typ-Funktion. Danach folgen die Suchergebnisse mit den gefundenen Benutzernamen und an letzter Stelle die Suchergebnisse der Namen der für den Benutzer sichtbaren Topic Maps.

5.2.3 *Die Topic-Map-Suche*

Die Topic-Map-Suche konzentriert sich auf eine einzelne Topic Map. Analog zum Vorgehen bei der Seitensuche wird die Suchanfrage des Benutzers an eine neue Instanz des Rails-Modells `Search` übergeben, welche die Suchanfrage an eine neue Instanz der Klasse `RTM::Search::Searcher` weiterleitet.

Neben dem semantischen Volltextindex der Topic Map wird bei der Topic-Map-Suche auch auf den beim Indexieren angelegten Facettenindex zurückgegriffen. Da für jedes Topic eine generische Facette mit dem jeweiligen Topic-Typ angelegt wird, besteht die Möglichkeit, die Liste der Suchergebnisse mit einem Facettenfilter einzuschränken.

Dazu werden die in den Suchergebnissen vorkommenden Topic-Typ-Facetten über den zunächst vollständigen Suchergebnissen angezeigt. Durch eine Abwahl der gewünschten Markierungsfelder kann die Menge der Suchergebnisse eingeschränkt werden (vgl. Abbildung 11). Bei inkonsistent modellierten Topic Maps kann zusätzlich der Fall auftreten, dass ein Topic keinen Topic-Typ besitzt. Daher gibt es neben den Markierungsfeldern für die gefundenen Topic-Typ auch ein Feld für Topic-Ergebnisse ohne Topic-Typen.

6 ZUSAMMENFASSUNG

Dieses Kapitel enthält eine Zusammenfassung der in dieser Arbeit untersuchten Themen und eine Einschätzung, inwieweit die anfänglich formulierten Ziele erreicht werden konnten.

6.1 SEMANTISCHE SUCHE IN TOPIC MAPS

Ein Ziel dieser Arbeit war die Entwicklung einer semantischen Suchmaschine, welche bei allen Verarbeitungsschritten ein besonderes Augenmerk auf die Möglichkeiten legt, welche sich aus einer Topic-Maps-Datenbasis ergeben. Die Forderung nach einer möglichst breit gefächerten Verwendbarkeit der Suchmaschine, auch in bereits vorhandener Topic-Maps-Software, konnte durch das Aufsetzen auf das in Kapitel 2 vorgestellte Topic Maps Application Programming Interface (TMAPI) erfüllt werden.

In Kapitel 3 wurden zunächst verschiedene auf Topic-Maps-Daten spezialisierte Indexierungsalgorithmen anhand einiger Bewertungskriterien untersucht. Der für die Implementierung ausgewählte Topic-zentrierte Ansatz zeigte sich hinsichtlich seiner Effizienz und der gebotenen Index-Qualität als überlegen. Dieser Ansatz erlaubt zudem ein unkompliziertes inkrementelles Reindexieren, sodass die Index-Qualität auch bei nachträglichen Änderungen an der Datenbasis nicht sinkt.

Einen Spezialfall stellt die Indexierung virtuell zusammengeführter Topic Maps dar. Zu diesem Problem wurden verschiedene Lösungsmöglichkeiten untersucht. Es konnte gezeigt werden, dass die Indexierung virtuell zusammengeführter Topic Maps mit dem gewählten Ansatz ohne weiteres möglich ist.

Es wurden verschiedene Suchmaschinenbibliotheken auf ihre Anwendbarkeit hin untersucht, wobei die Bibliothek *Apache Lucene* als am besten geeignet identifiziert werden konnte. Mit dieser Spezialbibliothek kann eine hohe Effizienz bei der Erstellung, Verwaltung und Durchsuchung eines Indexes gewährleistet werden.

Darüber hinaus bietet Lucene dem Benutzer eine große Flexibilität bei der Formulierung seiner Suchanfrage. Dies wird durch Lucenes umfangreiche Anfragesprache ermöglicht, welche sowohl simple als auch komplexe Suchanfragen ermöglicht. Das sorgt für eine hohe Recherchequalität der Suchmaschine.

Durch die Einbeziehung der vorhandenen Topic-Maps-immanenten Elemente mit impliziter Bedeutung, sowohl bei der Indexierung als auch bei der Gewichtung der Suchergebnisse, kann eine hohe Trefferqualität gewährleistet werden. Sucht ein Nutzer

nun beispielsweise in der Datenbasis aus Abschnitt 2.1.2 nach dem Wort „Kindergarten“, so finden sich unter den Suchergebnissen nicht nur diese Topics, welche die Zeichenkette enthalten, sondern auch alle Topics, die Instanzen des Topic-Typs mit dem Namen „Kindergarten“ sind. Damit wird dem eigentlichen Informationsbedürfnis des Nutzers wesentlich stärker Rechnung getragen.

6.2 FACETTIERTE SUCHE IN TOPIC MAPS

Ein weiteres Ziel dieser Arbeit war die Bereitstellung von Funktionen, die dem Nutzer eine explorative Suche ermöglichen. Um dieses Ziel zu erreichen, wurde ein allgemeines Modell zur Beschreibung von Topic-Maps-basierten Facetten vorgestellt. Darauf aufbauend wurden Möglichkeiten der Erstellung generischer Facetten aufgezeigt. Weiterhin wurde mit Hilfe der Topic-Maps-Abfragesprache TMQL eine Methode zur Definition von domänen-spezifischen Facetten entworfen und erläutert.

Um die große Anzahl erstellter Facetten automatisch ordnen zu können, wurden mehrere Metriken behandelt, die sich für diesen Zweck sehr gut eigenen und mit denen eine automatische Ordnung berechnet werden kann.

Die Speicherung der generierten Facetten erfolgt, analog zu den semantischen Volltextinformationen, in einem Lucene-Index. Damit verfügt der Nutzer auch bei der Abfrage der Facetten über die von Lucene offerierten Vorteile bezüglich Effizienz und Flexibilität bei der Formulierung der Anfrage. Darüber hinaus kann durch die Implementierung von Methoden für inkrementelles Reindexieren eine gleichbleibend hohe Index-Qualität gewährleistet werden.

6.3 INTEGRATION DER SUCHE IN WEBAPPLIKATIONEN

Das dritte Ziel dieser Arbeit bestand in einer prototypischen Implementierung einer Schnittstelle, mit der die entstandene Suchmaschine in Topic-Maps basierten Webapplikationen genutzt werden kann. Dies wurde durch die Schaffung eines neuen Pakets für die Middleware RTM ermöglicht, welches die entstandene Java-Bibliothek in JRuby nutzbar macht, sodass die Suchmaschine unkompliziert in Ruby-on-Rails-Webapplikationen genutzt werden kann.

Aufbauend auf diesem Modul wurde die Suchmaschine in den Topic-Maps-Browser Maiana integriert, in dem sie an zwei verschiedenen Stellen zum Einsatz kommt. Mit diesem praktischen Anwendungsfall konnte die Leistungsfähigkeit der einzelnen Komponenten der Suchmaschine – semantischer Index, flexible Abfrage und generische Facetten – gezeigt werden.

A

TERMINOLOGIE

Die in dieser Arbeit verwendete Terminologie zur Beschreibung von Topic Maps orientiert sich an der von Maicher in [26] entwickelten deutschsprachigen Topic-Maps-Terminologie. Zusätzlich zu den dort definierten Begriffen werden einige Begriffe neu eingeführt. Ihre Bedeutung und Herkunft wird in Tabelle 7 erläutert.

Zur Schreibweise zusammengesetzter Begriffe definiert der Duden in §§43 – 44 die Durchkopplung. Diese sieht vor, dass zusammengesetzte Begriffe mit Bindestrich geschrieben werden. In der deutschen Topic-Maps-Terminologie besteht als Ausnahme davon der Begriff „Topic Map“.

DEUTSCH	ENGLISCH (HERKUNFT)
Typ-Instanz-Beziehung	type-instance relationship (TMDM [19])
Basisadresse	base locator (TMAPI 2.0.2 [9])

Tabelle 7: Ergänzungen zur Terminologie

B

ÜBERSICHT DER INDEXIERTEN FELDER

Dieses Kapitel enthält eine umfassende Übersicht über die Eigenschaften der Felder, die ein mit der Klasse `TopicMapIndexer` erzeugter Index enthält. Alle Beispiele beziehen sich auf die in Abschnitt 3.2.3 beschriebene Topic Map *toyTM*.

B.1 FELDER FÜR IDENTIFIZIERENDE ELEMENTE

Die Anführungszeichen sind bei URIs notwendig, da der Doppelpunkt den Feldnamen vom Suchbegriff trennt und damit ein reserviertes Zeichen ist (vgl. Abschnitt 3.3.3).

B.1.1 *Das Feld si*

Das Feld `si` dient zur Speicherung der indirekten Gegenstandsanzeiger eines Topics. Das Basisgewicht ist 1,0.

Beispiel: `si:"http://en.wikipedia.org/wiki/City"`
Findet alle Topics, die diesen indirekten Gegenstandsanzeiger tragen.

B.1.2 *Das Feld sl*

Das Feld `sl` dient zur Indexierung der direkten Gegenstandsanzeiger eines Topics. Das Basisgewicht beträgt 1,0.

Beispiel: `sl:"http://en.wikipedia.org/wiki/Germany"`
Findet die Topics mit diesem direkten Gegenstandsanzeiger.

B.2 FELDER FÜR ELEMENTE MIT EXPLIZITER BEDEUTUNG

B.2.1 *Das Feld name*

Im Feld `name` werden die Werte der Benennungen eines Topics mit dem Basisgewicht 1,5 gespeichert. Werte, die aus Namensvarianten stammen, erhalten das Basisgewicht 1,0.

Beispiel: `name:Germany`
Findet alle Topics, die *Germany* in einer Benennung tragen.

B.2.2 *Das Feld occurrence*

Das Feld `occurrence` enthält die Werte der Belegstellen eines Topics. Das Basisgewicht ist 1,0.

Beispiel: `occurrence:Uta`
Findet alle Topics, die „Uta“ in einer Belegstelle enthalten.

B.3 FELDER FÜR ELEMENTE MIT IMPLIZITER BEDEUTUNG

B.3.1 *Das Feld type*

Felder des Typs `type` tragen Informationen über die Topic-Typen des indexierten Topics mit dem Basisgewicht 2,0.

Beispiel: `type:Kingdom`
Findet alle Instanzen des Topic-Typs „Kingdom“.

B.3.2 *Das Feld role*

Im Feld `role` werden die Namen der Rollen hinterlegt, die ein Topic spielt, wobei das Basisgewicht 1,0 beträgt.

Beispiel: `role:Container`
Findet alle Topics, die die Rolle „Container“ spielen.

B.3.3 *Das Feld assoc_type*

Das Feld `assoc_type` enthält die Namen der Beziehungstypen, an denen ein Topic teilnimmt. Das Basisgewicht ist 1,0.

Beispiel: `assoc_type:Capital`
Findet alle Topics, die an einer Beziehung teilnehmen, die den Typ „Capital“ hat.

B.3.4 *Das Feld assoc_player*

Im Feld `assoc_player` werden die Namen der Topics indexiert, mit denen das zu indexierende Topic gemeinsam an einer Beziehung teilnimmt. Das Basisgewicht beträgt 1,0.

Beispiel: `assoc_player:Germany`
Findet alle Topics, die an einer Beziehung zu einem Topic namens „Germany“ teilnehmen.

B.3.5 *Das Feld is_type*

Das Feld `is_type` zeigt an, ob das indexierte Topic als Topic-Typ an einer Typ-Instanz-Beziehung teilnimmt, wobei das Basisgewicht 1,0 beträgt.

Beispiel: `is_type:true`
Diese Suche liefert alle Topic-Typen, die Instanzen besitzen. Topics, die zwar Subtypen, aber keine Instanzen besitzen, werden nicht gefunden.

C

KONFIGURATION DER KLASSEN

Dieses Kapitel enthält Informationen zu den Konfigurationsmöglichkeiten der einzelnen Klassen. Die Konfiguration kann sowohl über die Konfigurationsdateien der einzelnen Klassen als auch über die jeweiligen Abfrage bzw. Änderungsmethoden erfolgen.

C.1 DIE KLASSE TOPICMAPINDEXER

C.1.1 *Überschreiben eines vorhandenen Index*

Die Methode `index()` testet vor dem Anlegen eines Index, ob bereits ein Index für die Basisadresse der übergebenen Topic Map existiert. Fällt dieser Test positiv aus, wird der vorhandene Index nicht überschrieben. Wird dagegen die Eigenschaft `allowOverwrite` auf den Wert `true` gesetzt, so wird der eventuell vorhandene Index mit einem neuen überschrieben.

C.1.2 *Automatisches Facettieren*

Wird die Eigenschaft `createFacets` auf den Wert `true` gesetzt, so wird bei jedem Aufruf der Methode `index()` automatisch der passende Facettenindex angelegt. Da dies eine gewisse Zeit benötigt, ist der Standardwert dieser Eigenschaft `false`.

C.1.3 *Indexieren identifizierender Elemente*

Für jeden Index kann angegeben werden, welche identifizierenden Elemente indexiert werden sollen. Dazu dienen die folgenden Eigenschaften:

- `indexSubjectIdentifier` für indirekte Gegenstandsanzeiger. Standardwert: `true`
- `indexSubjectLocator` für direkte Gegenstandsanzeiger. Standardwert: `true`
- `indexItemIdentifier` für Identifizierungen. Standardwert: `true`
- `indexTopicId` für die interne Adresse eines Topic-Maps-Konstrukts. Standardwert: `false`

C.1.4 *Sprache der Grundformreduktion*

Soll beim Indexieren eine Grundformreduktion durchgeführt werden, so muss die Eigenschaft `stemmerLanguage` entsprechend konfiguriert werden[1]. Enthält diese Eigenschaft keinen Wert, so findet keine Grundformreduktion statt. Wird dagegen eine nicht verfügbare Sprache konfiguriert, so wird das Indexieren nicht funktionieren. Weiterhin muss darauf geachtet werden, dass die Einstellungen für das Indexieren und das Suchen exakt übereinstimmen.

C.2 DIE KLASSE TOPICMAPFACETTER

Da die Klasse `TopicMapFacetter` von der Klasse `TopicMapIndexer` abgeleitet wurde, enthält sie die gleichen Konfigurationsmöglichkeiten. Da allerdings weder die Identifizierung eines Topics noch die interne Adresse eines Topic-Map-Konstrukts für die Identifizierung eines facettierten Instanztopics geeignet sind, werden die Eigenschaften `indexItemIdentifier` und `indexTopicId` immer auf den Wert `false` gesetzt.

C.3 DIE KLASSE TOPICMAPSEARCHER

C.3.1 *Führende Platzhalter*

Aufgrund der optimierten Struktur des invertierten Index von Lucene führt die Verwendung von führenden Platzhaltern bei der Suche zu Geschwindigkeitseinbußen. Um dieses Merkmal nutzen zu können, muss die Eigenschaft `allowLeadingWildcard` den Wert `true` erhalten, da der Standardwert `false` ist.

C.3.2 *Automatische Suche nach Facetten*

Wird der Wert der Eigenschaft `searchFacets` auf `true` gesetzt, so werden für alle Ergebnisse der Methode `search()` automatisch die entsprechenden Facetten im Facettenindex gesucht. Dieser Vorgang verzögert allerdings die Rückgabe der Suchergebnisse, weshalb der Standardwert dieser Eigenschaft `false` ist.

C.3.3 *Suche nach identifizierenden Elementen*

Mit den folgenden Eigenschaften kann für die verschiedenen identifizierenden Elemente angegeben werden, ob die jeweiligen Felder des Index durchsucht werden sollen.

1 Informationen über die verfügbaren Sprachen und die verwendeten Algorithmen sind unter http://snowball.tartarus.org zu finden.

- `searchSubjectIdentifier` für die Suche nach indirekten Gegenstandsanzeigern. Standardwert: `true`
- `searchSubjectLocator` für die Suche nach direkten Gegenstandsanzeigern. Standardwert: `true`
- `searchItemIdentifier` für die Suche nach Identifizierungen. Standardwert: `false`
- `searchTopicId` für die Suche nach den internen Adressen der Topic-Maps-Konstrukte. Standardwert:`false`

C.3.4 *Maximale Anzahl von Suchergebnissen*

Der Standardwert für die Eigenschaft `maxSearchResults` beträgt `1000`. Sollten mehr Suchergebnisse möglich sein, so muss dieser Wert entsprechend angepasst werden.

C.3.5 *Sprache der Grundformreduktion*

Soll bei der Suche eine Grundformreduktion durchgeführt werden, so muss die Eigenschaft `stemmerLanguage` entsprechend konfiguriert werden. Enthält diese Eigenschaft keinen Wert, so findet keine Grundformreduktion statt. Wird dagegen eine nicht verfügbare Sprache konfiguriert, so wird die Suche nicht funktionieren. Weiterhin muss darauf geachtet werden, dass die Einstellungen für das Indexieren und das Suchen exakt übereinstimmen.

LITERATURVERZEICHNIS

[1] Baeza-Yates, R.; Ribeiro-Neto, B.: *Modern Information Retrieval*. Addison Wesley Longman Ltd., Essex, Großbritannien, 1999 (Zitiert auf den Seiten 18 und 34)

[2] Blair, David C.; Maron, M. E.: An evaluation of retrieval effectiveness for a full-text document-retrieval system. In: *Commun. ACM* 28 (1985), Nr. 3, S. 289–299 (Zitiert auf Seite 7)

[3] Bleier, A.; Bock, B.; Schulze, U.; Maicher, L.: JRuby Topic Maps. In: Maicher, L.; Garshol, L.M. (Hrsg.): *Linked Topic Maps. Fifth International Conference on Topic Maps Research and Applications, TMRA 2009*. LIV, Leipzig, 2009, S. 195–205 (Zitiert auf den Seiten 13 und 45)

[4] Bock, B.: *Topic-Maps-Middleware. Modellgetriebene Entwicklung kombinierbarer domänenspezifischer Topic-Maps-Komponenten*, Universität Leipzig, Diplomarbeit, 2008 (Zitiert auf Seite 13)

[5] Broder, A.: A Taxonomy of Web Search. In: *ACM SIGIR Forum* 36 (2002), Nr. 2, S. 3–10 (Zitiert auf Seite 29)

[6] Damerau, F. J.: A technique for computer detection and correction of spelling errors. In: *Communications of the ACM* 7 (1964), Nr. 3, S. 171–176 (Zitiert auf Seite 30)

[7] Delbru, R.: Manipulation and Exploration of Semantic Web Knowledge / DERI and EPITA. 2006. – Internship Report (Zitiert auf Seite 40)

[8] Eastman, C. M.; Jansen, B. J.: Coverage, Relevance, and Ranking: The Impact of Query Operators on Web Search Engine Results. In: *ACM Transactions on Information Systems* 21 (2003), Nr. 4, S. 383–411 (Zitiert auf Seite 31)

[9] Espen, H.; Schmidt, J.: *PHPTMAPI 2.0*. 2010. – http://phptmapi.sourceforge.net/ (Zitiert auf den Seiten 13 und 53)

[10] Garshol, L.M.: A citizen's Portal for the City of Bergen. In: Maicher, L.; Garshol, L.M. (Hrsg.): *Scaling Topic Maps. Third International Conference on Topic Maps Research and Applications, TMRA 2007*. LIV, Leipzig, 2007, S. 23–35 (Zitiert auf Seite 7)

[11] GARSHOL, L.M.: Towards Semantic Search with Topic Maps. In: MAICHER, L.; GARSHOL, L.M. (Hrsg.): *Linked Topic Maps. Fifth International Conference on Topic Maps Research and Applications, TMRA 2009*. LIV, Leipzig, 2009, S. 107–114 (Zitiert auf Seite 8)

[12] HEIM, P.; ERTL, T.; ZIEGLER, J.: Facet Graphs: Complex Semantic Querying Made Easy. In: *The Semantic Web: Research and Applications* Bd. 6088, Springer, 2010 (LNCS), S. 288–302 (Zitiert auf den Seiten 1 und 28)

[13] HEUER, L.; SCHMIDT, J.: TMAPI 2.0. In: MAICHER, L.; GARSHOL, L.M. (Hrsg.): *Subject-centric computing. Fourth International Conference on Topic Maps Research and Applications, TMRA 2008*. LIV, Leipzig, 2008, S. 129–136 (Zitiert auf Seite 13)

[14] HEYER, G.; QUASTHOFF, U.; WITTIG, T.: *Text Mining: Wissensrohstoff Text*. 1. Auflage. W3l, Witten, Deutschland, 2006 (Zitiert auf den Seiten 15, 20 und 30)

[15] ISO/IEC 13250-3: *Topic Maps – XML Syntax (XTM)*. International Organization for Standardization, Genf, Schweiz, 2006. – http://www.isotopicmaps.org/sam/sam-xtm/ (Zitiert auf Seite 5)

[16] ISO/IEC 13250-5: *Topic Maps – Reference Model (TMRM)*. International Organization for Standardization, Genf, Schweiz, 2007. – http://www.isotopicmaps.org/tmrm/ (Zitiert auf Seite 4)

[17] ISO/IEC 13250-6: *Topic Maps – Compact Syntax (CTM)*. International Organization for Standardization, Genf, Schweiz, 2006. – http://www.isotopicmaps.org/ctm/ (Zitiert auf Seite 5)

[18] ISO/IEC 18048: *Topic Maps – Query Language (TMQL)*. International Organization for Standardization, Genf, Schweiz, 2008. – http://www.isotopicmaps.org/tmql/tmql.html (Zitiert auf den Seiten 12, 13 und 41)

[19] ISO/IEC WD 13250-2: *Topic Maps – Data Model (TMDM)*. International Organization for Standardization, Genf, Schweiz, 2008. – http://www.isotopicmaps.org/sam/sam-model/ (Zitiert auf den Seiten 4, 12, 22 und 53)

[20] KROSSE, S.: *Konzeption, Implementierung und Evaluierung eines TMQL-Parsers und Interpreters*, Universität Leipzig, Masterarbeit, 2009 (Zitiert auf Seite 12)

[21] LADWIG, G.; TRAN, T.: Combining Query Translation with Query Answering for Efficient Keyword Search. In: *The*

Semantic Web: Research and Applications Bd. 6089, Springer, 2010 (LNCS), S. 288–303 (Zitiert auf Seite 29)

[22] LEWANDOWSKI, D.: *Web Information Retrieval – Technologien zur Informationssuche im Internet*. Deutsche Gesellschaft für Informationswissenschaft und Informationspraxis e. V., Frankfurt am Main, 2005 (Zitiert auf den Seiten 1, 14 und 44)

[23] LEWANDOWSKI, D.: Zur Bewertung der Qualität von Suchmaschinen. In: EBERSPÄCHER, J.; HOLTEL, S. (Hrsg.): *Suchen und Finden im Internet*. 1. Auflage. Springer, Berlin/Heidelberg, 2006 (Zitiert auf Seite 15)

[24] LEWANDOWSKI, D.; HÖCHSTÖTTER, N.: Qualitätsmessung bei Suchmaschinen – System- und nutzerbezogene Evaluationsmaße. In: *Informatik Spektrum* 30 (2007), Nr. 3, S. 159–169 (Zitiert auf den Seiten 7 und 15)

[25] MAICHER, L.: *Autonome Topic Maps. Zur dezentralen Erstellung von implizit und explizit vernetzten Topic Maps in semantisch heterogenen Umgebungen*, Universität Leipzig, Diss., 2007 (Zitiert auf Seite 4)

[26] MAICHER, L.: *Deutsche Topic Maps Terminologie*. 2008. – http://www.informatik.uni-leipzig.de/~maicher/tmt/TMT.html (Zitiert auf Seite 53)

[27] MCCANDLESS, M.; HATCHER, E.; GOSPODNETIĆ, O.: *Lucene in Action*. 2. Auflage. Manning Publications, Greenwich, CT, USA, 2010 (Zitiert auf den Seiten 17, 18, 20 und 34)

[28] OREN, E.; DELBRU, R.; DECKER, S.: Extending Faceted Navigation for RDF Data. In: *The Semantic Web – ISWC 2006*. Springer, Berlin/Heidelberg, 2006, S. 559–572 (Zitiert auf den Seiten 37 und 40)

[29] PEPPER, S.: The TAO of Topic Maps. In: *Proceedings of XML Europe 2000*, 2000 (Zitiert auf Seite 36)

[30] PERUGINI, S.: Supporting multiple paths to objects in information hierarchies: Faceted classification, faceted search, and symbolic links. In: *Information Processing & Management* 46 (2010), Nr. 1, S. 22–43 (Zitiert auf Seite 42)

[31] PORTER, M. F.: An algorithm for suffix stripping. In: *Program* 14 (1980), Nr. 3, S. 130–137 (Zitiert auf den Seiten 20 und 30)

[32] RAJARAMAN, A.: Kosmix: High-Performance Topic Exploration using the Deep Web. In: *Proceedings of the VLDB Endowment* Bd. 2. VLDB Endowment, Mountain View, CA, USA, 2009, S. 1524–1529 (Zitiert auf Seite 29)

[33] SACCO, G.M.; TZITZIKAS, Y. (Hrsg.): *Dynamic Taxonomies and Faceted Search*. Springer, Berlin/Heidelberg, 2009 (Zitiert auf Seite 36)

[34] SCHULZE, U.: *Maiana – Share, mix and explore your data interlinked*. 2010. – http://www.topicmapslab.de/publications/lswt2010 (Zitiert auf Seite 47)

[35] SMOLNIK, S.: Convergence of Classical Search and Semantic Technologies – Evidences from a Practical Case in the Chemical Industry. In: MAICHER, L.; GARSHOL, L.M. (Hrsg.): *Scaling Topic Maps. Third International Conference on Topic Maps Research and Applications, TMRA 2007*. LIV, Leipzig, 2007, S. 14–24 (Zitiert auf Seite 33)

[36] TAYLOR, Arlene G.: *Introduction to cataloging and classification*. Englewood, Colorado, 2000 (Zitiert auf Seite 36)

[37] TOPICMAPS.ORG AUTHORING GROUP, Members of t.; PEPPER, S.; MOORE, G. (Hrsg.): *XML Topic Maps (XTM) 1.0*. TopicMaps.Org, 2001. – http://www.topicmaps.org/xtm/ (Zitiert auf Seite 37)

[38] TRAN, T.; MATHÄSS, T.; HAASE, P.: Usability of Keyword-Driven Schema-Agnostic Search: A Comparative Study of Keyword Search, Faceted Search, Query Completion and Result Completion. In: *The Semantic Web: Research and Applications* Bd. 6089, Springer, 2010 (LNCS), S. 349–364 (Zitiert auf den Seiten 1, 28, 36 und 41)

[39] UEBERALL, M.; DROBNIK, O.: Facet-based Exploratory Search in Topic Maps. In: MAICHER, L.; GARSHOL, L.M. (Hrsg.): *Subject-centric computing. Fourth International Conference on Topic Maps Research and Applications, TMRA 2008*. LIV, Leipzig, 2008, S. 49–62 (Zitiert auf Seite 39)

ABBILDUNGSVERZEICHNIS

Abbildung 1 Beispiel einer klassischen Auflistung von Suchergebnissen 2
Abbildung 2 Schematische Darstellung aller Elemente des TMDM, die bei der Bildung einer (binären) Beziehung beteiligt sind. 9
Abbildung 3 Die Komponenten einer Suchmaschine. . . . 14
Abbildung 4 Übersicht über die von Lucene zur Verfügung gestellten Funktionen. 17
Abbildung 5 Schema der Indexierung einer a priori virtuell zusammengeführten Topic Map. 27
Abbildung 6 Schema der Indexierung einer nicht a priori virtuell zusammengeführten Topic Map. . . 28
Abbildung 7 Beispielmodell mit drei Topic-Typen: Person, Adresse und Stadt. 41
Abbildung 8 Die verschiedenen Komponenten von JRTM. 45
Abbildung 9 Hauptansicht einer Topic Map in Maiana mit den zwei Eingabefeldern der Suche. . . 48
Abbildung 10 Ergebnisse der Seitensuche mit dem Suchbegriff *toytm*. 49
Abbildung 11 Ergebnisse der Topic-Map-Suche mit Topic-Typ-Facetten, mit Einschränkung auf Topics vom Typ „Republic“ und „Kingdom“ 50

TABELLENVERZEICHNIS

Tabelle 1 Evaluierte Suchbibliotheken 16
Tabelle 2 Ergebnisse der Evaluation der verschiedenen Indexierungsalgorithmen. 23
Tabelle 3 Übersicht der Topic-Map-Konstrukte, ihrer Verwendung im Volltextindex und ihrer Basisgewichte 25
Tabelle 4 Update-Typen und ihre Bedeutung 26
Tabelle 5 Die Ergebnistypen und ihre Informationen . 33
Tabelle 6 Übersicht der für die Facettierung eines Topics genutzten Topic-Map-Konstrukte 39
Tabelle 7 Ergänzungen zur Terminologie 53

ERKLÄRUNG

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

Leipzig, Oktober 2010

Sven Windisch B. Sc.