Entwicklung einer Suchfunktion mit Information-Retrieval-Schnittstelle für den Dokumentenserver OPUS unter besonderer Berücksichtigung von Entwurfsmustern

BACHELOR REPORT

zur Erlangung des akademischen Grades
Bachelor of Science in Computer Science
(B.Sc.)
im Fach Informatik

eingereicht an der
Virtuelle Fachhochschule / Fachhochschule Lübeck
Fachbereich Elektrotechnik

von
Dipl.-Bibl. (FH) Oliver Christian Marahrens

URN: urn:nbn:de:gbv:830-tubdok-3572
Gutachter:

1. Prof. Dr. Silke Seehusen
2. Prof. Dr. Holger Hinrichs

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodelistingverzeichnis

Abkürzungsverzeichnis

API ..... Application Programming Interface Schnittstelle, über die Funktionalitäten zur Anbindung eines Programmes auf Quelltextebene zur Verfügung gestellt werden. Vgl. [Wik07a]

ASF ..... Atom Syndication Format XML-basiertes Format als Erweiterung von RSS. `http://www.atompub.org/`

BSZ ..... Bibliotheksservice-Zentrum Baden-Würtemberg Für den SWB zuständiger bibliothekarischer Dienstleister. `http://www.bsz-bw.de`

CC ..... Creative Commons Lizenzmodell zum Zusammenstellen freier Lizenzen für geistiges Eigentum. `http://creativecommons.org/`

CCS ..... Computing Classification System Klassifikationssystem für Informatik-Titel. `http://www.acm.org/class/1998/TOP.html`

CGI ..... Common Gateway Interface Standard für den Datenaustausch zwischen Webserver und externem Programm, wird oft zur Erstellung dynamischer Webseiten benutzt, indem das aufgerufene Programm HTML-Code an den Webserver zurückgibt. `http://de.wikipedia.org/wiki/Common_Gateway_Interface`

DC ..... Dublin Core Von der DCMI entwickeltes Metadatenformat. `http://dublincore.org/documents/dcmi-terms/`

DDC ..... Dewey-Decimal-Classification Ein altes, aber nach wie vor in Bibliotheken sehr verbreitetes Universalklassifikationssystem zur groben Kategorisierung von Medien. `http://www.ddc-deutsch.de/`

DFN ..... Deutsches Forschungsgsnetz Kommunikationsnetz zwischen Hochschulen in Deutschland. `http://www.dfn.de`

DIMDI ..... Deutsches Institut für Medizinische Dokumentation und Information Kommerzieller Datenbankhost. http://www.dimdi.de/

DINI ..... Deutsche Initiative für Netzwerkinformation Verein zur Koordination des Umgangs mit modernen Informations- und Kommunikationsmethoden an deutschen Hochschulen und Forschungseinrichtungen. http://www.dini.de

DRM ..... Digital Rights Management Verfahren zur Kontrolle der Nutzung elektronischer Ressourcen. http://de.wikipedia.org/wiki/Digitale_Rechteverwaltung

GPG ..... Gnu Privacy Guard System zur Verschlüsselung von Daten sowie zum digitalen Signieren von Dokumenten. http://www.gnupg.org

GPL ..... GNU General Public Licence Verbreitete Softwarelizenz für OpenSource-Software. http://www.gnu.org/licenses/

HTML ..... HyperText Markup Language Format zur Darstellung von Webseiten im Browser. http://www.w3.org/TR/html4/

HTTP ..... HyperText Transfer Protocol Vor allem im Internet verwendetes Textübertragungsprotokoll.

KOBV ..... Kooperativer Bibliotheksverbund Bibliotheksverbund für die Region Berlin-Brandenburg. http://www.kobv.de/

MSC ..... Mathematics Subject Classification Klassifikationssystem für Mathematik-Titel. http://www.ams.org/msc/

OA ..... OpenAccess Open Access bedeutet die kosten- und hindernisfreie Zugänglichmachung wissenschaftlicher Online-Publikationen und -Materialien. http://de.wikipedia.org/wiki/Open_Access

OAI ..... Open Archives Initiative Die Open Archives Initiative entwickelt und vermarktet Standards zum Aufbau von Interoperabilität, die dazu dienen, Inhalte effizient verteilen zu können.[1] http://www.openarchives.org/

OAI-PMH .... Open Archives Initiative - Protocol for Metadata Harvesting Von der OAI entwickeltes, XML-basiertes Protokoll zum Austausch von Metadaten. http://www.openarchives.org/pmh/

[1] übersetzt aus [OAI07]

OPUS ..... Online Publikatonssystem der Universität Stuttgart OPUS ist ein Dokumentenserversystem, das an der Universitätsbibliothek der Universität Stuttgart entwickelt wurde. http://elib.uni-stuttgart.de/opus/doku/about.php

PACS ..... Physics and Astronomy Classification Scheme Klassifikationssystem für Titel aus den Fachgebieten Physik und Astronomie. http://publish.aps.org/PACS/pacsgen.htm

PDF ..... Portable Document Format Proprietäres, aber offengelegtes Dokumentformat der Firma Adobe. http://www.adobe.com/products/acrobat/adobepdf.html

PHP ..... PHP Hypertext Preprocessor Scriptsprache, hauptsächlich zur Programmierung dynamischer Webseiten und Webapplikationen verwendet. http://www.php.net

PS ..... PostScript PostScript ist eine Seitenauszeichnungssprache und wurde wie PDF von der Firma Adobe entwickelt. PostScript-Dokumente können genau so wie sie sind auf einem postscriptfähigen Drucker ausgegeben werden. Vgl. [Wik07e]

RSS ..... Really Simple Syndication XML-basiertes Format zum Austausch von Metadaten zwischen Diensteanbietern im Web. http://www.rssboard.org/rss-2-0

SEO ..... Search Engine Optimizing Maßnahmen, um den eigenen Datenbestand in Suchmaschinen sichtbar zu machen.

SQL ..... Structured Query Language Standard-Abfragesprache zum Durchsuchen und Manipulieren von Daten in relationalen Datenbanken. http://de.wikipedia.org/wiki/SQL

SWB ..... Süd-West-Verbund Südwestdeutschen Bibliotheksverbund für Baden-Württemberg, Saarland und Sachsen. http://titan.bsz-bw.de/cms/

TUBHH ..... Universitätsbibliothek der Technischen Universität Hamburg-Harburg http://www.tub.tu-harburg.de

UML ..... Unified Modeling Language Standard zur graphischen Darstellung und Beschreibung der Struktur objektorientierter Softwaresysteme. http://uml.org/

URL ..... Uniform Resource Locator Adresse einer elektronischen Ressource im Internet.

URN ..... Uniform Resource Name Persistent Identifier, mit dem eine elektronische Ressource unabhängig von ihrem URL im Internet identifiziert werden kann. In Deutschland übernimmt die Deutsche Nationalbibliothek die Koordination der URN-Vergabe und stellt einen Resolver zur Auflösung von URNs zu URLs zur Verfügung. `http://nbn-resolving.de/ResolverDemo.php`

XHTML ..... eXtensible HyperText Markup Language XML-konforme Erweiterung von HTML. `http://www.w3.org/TR/xhtml1/`

XML ..... eXtensible Markup Language Erweiterbare und anpassbare Markup-Sprache, die für viele Zwecke zu benutzen ist, z.B. Datenaustausch, Aufbau von Webseiten, Aufbau von Webservices, usw. `http://www.w3.org/XML/`

Glossar

Communities Ez.: Community. Gemeinschaft von Internetnutzern, die sich mit Hilfe verschiedener Kommunikationstools wie Chat, Instant Messaging oder Foren untereinander zu bestimmten Themen austauschen. Vgl. [Wik07d]

Connotea Auf wissenschaftliche Nutzung ausgelegter Social Bookmarking-Dienst und Reference-Manager des Journals Nature. `http://www.connotea.org`

del.icio.us Verbreiteter Social-Bookmarking-Dienst. `http://del.icio.us`

Dmoz Dmoz oder ODP (Open Directory Project) ist schon ein recht altes Web-Verzeichnis, das bereits 1998 von Netscape ins Leben gerufen wurde (vgl. Copyright-Hinweis auf `http://dmoz.org`). Das Verzeichnis ist offen und alle können teilnehmen, das Verzeichnis zu ergänzen und zu erweitern. `http://dmoz.org`

doc Das doc-Format ist ein binäres, proprietäres Format der Firma Microsoft, das ursprünglich mit dem Textverarbeitungsprogramm Word erzeugt wird.

Foren Ez.: Forum. Werkzeug zum Austausch mit anderen Internetnutzern. Meist sind Foren auf bestimmte Themenbereiche begegrenzt. Vgl. [Wik07c]

GBV-Basisklassifikation *Basisklassifikation des Gemeinsamen Bibliotheksverbundes* Universelles Klassifikationssystem, das im Bereich des norddeutschen Gemeinsamen Bibliotheksverbundes verwendet wird. `https://katalog.b.tu-harburg.de/DB=1/ADVANCED_SUBJECTS`

Index Unter einem Index wird im Rahmen dieser Arbeit die strukturierte und durchsuchbare Datenbasis einer Suchmaschine verstanden.

Luke Javaprogramm zum Durchsuchen, Überprüfen und Optimieren von Lucene-Indizes. `http://getopt.org/luke/`

MIME *Multipurpose Internet Mail Extensions* Ursprünglich aus dem Bereich der eMail stammend kann MIME auch zur Beschreibung von Formaten und Struk-

turen in anderen Kontexten verwendet werden. http://de.wikipedia.org/wiki/Multipurpose_Internet_Mail_Extensions

OCLC PICA Besonders in wissenschaftlichen Bibliotheken verbreitete Bibliothekssoftware. http://www.oclcpica.org/

odt *Open Document Text* ODT ist ein Format aus der ODF-Familie, das in erster Linie mit dem OpenOffice Writer erzeugt wird.

pdftotext Software zum Konvertieren von PDF-Dokumenten in reinen ASCII-Text. pdftotext wird über das Softwarepaket XPDF zur Verfügung gestellt. http://www.foolabs.com/xpdf/

Print-on-Demand Elektronische Publikation, die bei Bedarf durch einen darauf spezialisierten Anbieter gebunden und verkauft werden darf.

Recall Recall beschreibt die Vollständigkeit des Suchergebnisses. Die genaue Definition lautet nach Lewandowski: „Recall misst den Anteil der gefundenen relevanten Dokumente im Verhältnis zur Zahl der ingesamt im Datenbestand vorhandenen relevanten Dokumente."([Lew05], S. 139)

Reference Manager Online-Reference-Manager zählen zu Webdiensten im Rahmen des Web 2.0. Sie dienen dazu, Literaturangaben zu verwalten und mit anderen zu teilen. Ein bekannter Online-Reference-Manager ist Connotea. Dieser und ähnliche Dienste werden in der Web-2.0-Serie des Weblogs der Staats- und Universitätsbibliothek Hamburg vorgestellt. http://www.sub.uni-hamburg.de/blog/?p=433

Social Bookmarking Bookmarks sind Lesezeichen zu interessanten Webseiten. Das Prinzip des Social-Bookmarking überträgt das Bookmark-System aus Browsern in Online-Systeme. In Social-Bookmarking-Systemen können eigene Bookmarks auch anderen zur Verfügung gestellt werden. Ein bekannter Social-Bookmarking-Dienst ist del.icio.us. Dieser und ähnliche Dienste werden in der Web-2.0-Serie des Weblogs der Staats- und Universitätsbibliothek Hamburg vorgestellt. http://www.sub.uni-hamburg.de/blog/?p=388

SULB Saarbrücken *Saarländische Universitäts- und Landesbibliothek* http://www.sulb.uni-saarland.de/

TUBdok TUBdok ist der Name des Dokumentenservers der Technischen Universität Hamburg-Harburg, der von der Universitätsbibliothek betrieben wird. http://doku.b.tu-harburg.de

Web 2.0 Unter Web 2.0 werden Webseiten verstanden, die sich durch eine hohe Interaktivität auszeichenen. Der Nutzer selbst nimmt an der Gestaltung aktiv teil. Beispiele für Web 2.0 sind vor allem Wikis oder Weblogs.

Weblog Werkzeug zum einfachen und schnellen Veröffentlichen eigener Inhalte im Internet. Die Inhalte können je nach Konfiguration des Weblogs von anderen kommentiert werden. Vgl. [Wik07f]

Wiki Werkzeug zum einfachen und schnellen Veröffentlichen eigener Inhalte im Internet. Die Inhalte können je nach Wiki-Konfiguration und Zugänglichkeit von anderen Nutzern modifiziert oder geändert werden. Vgl. [Wik07g]

Zend-Framework Framework zur Erstellung von Applikationen mit PHP5. `http://framework.zend.com/`

Die Kunst des Suchens ist das Finden

Pablo Picasso

Kapitel 1

Einleitung

1.1 Ziel dieses Bachelor Reports

Dieser Bachelor Report beschreibt die Entwicklung einer Suchfunktion für den von vielen Universitäten bzw. Universitätsbibliotheken eingesetzten Dokumentenserver OPUS. Dabei finden die in der Suchmaschine verwendeten softwaretechnischen Entwurfsmuster besondere Berücksichtigung und werden dargestellt. Ziel der Suchmaschine ist, ein möglichst genaues Suchsystem abzubilden, das sowohl die Volltexte als auch die Metadaten umfasst und ein qualitativ hochwertiges Ranking bereitstellt.
Die Suchmaschine soll sowohl über eine Eingabemaske auf dem Dokumentenserver als auch über eine Retrieval-Schnittstelle zugänglich gemacht werden.

1.2 Arbeitsschritte zum Erreichen des Ziels

Zunächst muss eine geeignete Software oder Softwarebibliothek gefunden werden, die zur Implementierung der eigenen Suchmaschine verwendet werden kann. Für diese Suchmaschine muss eine Indexstruktur festgelegt werden, sofern diese nicht durch die benutzte Software vorgegeben ist. Es sind Klassen zu definieren, mit denen der Index gefüllt und anschließend durchsucht werden kann.
Für die Retrieval-Schnittstelle ist ein Austauschformat zu definieren, das dazu dient, Treffer zu einer Suchanfrage an beliebige Clients, die dieses Austauschformat interpretieren können, zu übermitteln.

1.3 Abgrenzung der Arbeit

In dieser Arbeit soll Suchmaschinentechnologie angewendet und im Kontext des universitären Dokumentenservers der TUBHH (Universitätsbibliothek der Techni-

schen Universität Hamburg-Harburg)[1] implementiert werden. Es geht jedoch nicht darum, neue Suchmaschinentechnologien zu entwickeln oder zu entwerfen. Alle Entwicklungen in dieser Arbeit basieren auf bereits vorhandenen Produkten, die jeweils ordnungsgemäß referenziert werden.
Die Dokumentenserverlandschaft in Deutschland und international basiert auf vielen verschiedenen Dokumentenserver-Produkten. In dieser Arbeit wird die Entwicklung nur auf einem dieser Produkte, dem OPUS-System, aufgesetzt. Als Entwicklungsumgebung dient dabei der OPUS-basierte Dokumentenserver TUBdok, der von der TUBHH betrieben wird. Das entstehende Suchsystem ist an TUBdok angepasst und kann in dieser Form nur in OPUS-basierten Systemen angewendet werden. Für die Retrieval-Schnittstelle gilt die Zielsetzung, sie möglichst universell einsetzbar zu halten, so dass sie bei Bedarf auch auf anderen Systemen implementiert werden kann.

1.4 Gliederung der Arbeit

Die Arbeit ist analog zu den in Abschnitt 1.2 beschriebenen Arbeitsschritten aufgebaut und gegliedert.

Grundlagen Zunächst werden im Kapitel 2 die zum Verständnis der Arbeit erforderlichen Grundlagen beschrieben und ausführlich dargestellt. Dabei wird insbesondere dargestellt, welche Informationsressourcen und Rechercheinstrumente es im Internet gibt und wie sie genutzt werden können. Es wird außerdem beschrieben, wie Dokumentenserver grundsätzlich funktionieren und dabei auf die beiden im Kontext dieser Arbeit relevanten Systeme OPUS und TUBdok eingegangen.

Suchmaschinen Backends In Kapitel 3 werden verschiedene Suchsysteme verglichen und auf ihre Eignung zur Verwendung in dieser Arbeit hin untersucht. Am Ende des Abschnitts wird eine Suchmaschine ausgewählt, die als Grundlage für die Eigenentwicklung dient.

Opus Suchsystem Die konkreten Anforderungen an das in dieser Arbeit entstehende Suchsystem werden in Kapitel 4 dargestellt. In dem Abschnitt wird außerdem die Modellierung des Systems beschrieben und die verwendeten Entwurfsmuster werden vorgestellt. Mit Hilfe von UML (Unified Modeling Language)-Diagrammen wird die Planung der Software dokumentiert.

[1] TUBdok, `http://doku.b.tu-harburg.de`

Information-Retrieval-Schnittstelle Kapitel 5 stellt die Anforderungen an die Information-Retrieval-Schnittstelle vor und beschreibt das verwendete Austauschformat.

Implementierung In Kapitel 6 wird die Umsetzung des Modells und die Integration in TUBdok dokumentiert. Es wird anhand von Quelltexten und Quelltext-Ausschnitten gezeigt, wie die Entwurfsmuster implementiert wurden.

Nachbereitung Nach der Implementierung des Systems folgen die Tests der Software, die mit Durchführung und Ergebnis in Kapitel 7 dokumentiert werden.

Abschluss Den Abschluss der Arbeit bildet das Kapitel 8, in dem die Arbeitsergebnisse zusammengefasst werden und ein Ausblick gegeben wird, was mit den Ergebnissen in der Zukunft passieren wird.

Kapitel 2

Grundlagen

2.1 Informationsrecherche im Internet

2.1.1 Informationsressourcen im Internet

Um im Internet Informationen zu finden, kann der Nutzer sich verschiedener Rechercheinstrumente bedienen. Stefan Karzauninkat unterscheidet drei verschiedene Typen von Suchdiensten (vgl. [Kar03], S. 22):

- Kataloge
- Roboterbasierte Suchmaschinen
- spezielle Suchdienste

Zur Gruppe der speziellen Suchdienste zählt Karzauninkat unter anderem Meta-Suchmaschinen ([Kar03], S. 102ff), Bibliothekskataloge ([Kar03], S. 127ff) und spezielle Suchtechniken wie Newsgroups oder Mailinglisten ([Kar03], S. 138ff). Unberücksichtigt lässt er jedoch Ressourcen wie Dokumentenserver, Datenbanken und RSS-Feeds.
Die Suchdienste stehen untereinander in Beziehung und das Geflecht zwischen ihnen ist sehr komplex und nahezu undurchschaubar (vgl. [Kar03], S. 162ff)[1]. In diesem Abschnitt werden Suchdienste, die im Kontext dieser Arbeit relevant sind, näher beschrieben.

Roboterbasierte Suchmaschinen

Nach Stefan Karzauninkat besteht „eine roboterbasierte Suchmaschine ... im Wesentlichen aus drei Teilen“ ([Kar03], S. 23). Der erste Teil ist der sogenannte Robot,

[1] Das Geflecht der Suchmaschinen ist genauer unter [Kar04] abgebildet - allerdings ist diese Abbildung von Ende 2004 inzwischen veraltet.

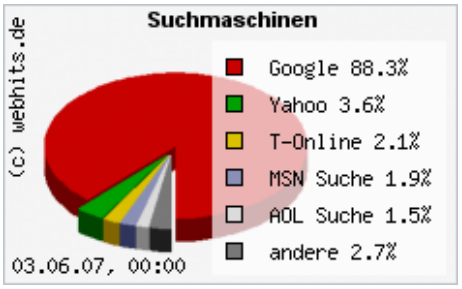

Abbildung 2.1: Marktanteile von Suchmaschinen in Deutschland Juni 2007 nach [Web07]

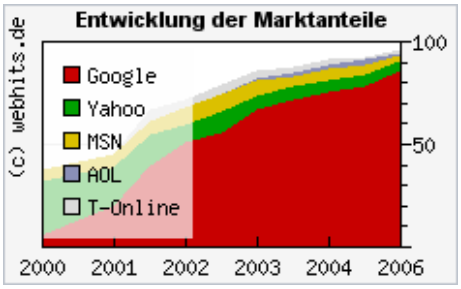

Abbildung 2.2: Entwicklung der Suchmaschinen-Marktanteile in Deutschland nach [Web07]

Spider oder Crawler, der sich ähnlich wie ein menschlicher Internetnutzer per Hyperlink durch das Web hangelt und auf seinem Weg die gefundenen Seiten in den Index der Suchmaschine aufnimmt (vgl. ebd.). Der zweite Teil ist nach Karzauninkat die Indizierungssoftware, die den Datenbestand der Suchmaschine hält und verwaltet. Der dritte Teil ist die Suchsoftware, die auf den Index aufsetzt und ihn durchsucht. Beispiele für robotbasierte Suchmaschinen sind Google[1], AltaVista[2] oder Yahoo[3].

Suchmaschinenmarkt Wie in den Abbildungen 2.1 und 2.2 ersichtlich, betrug der Marktanteil von Google in Deutschland Anfang Juni 2007 über 88%, Platz 2 geht an Yahoo mit 3,6%. Eine so einseitige Internetsuche birgt die Gefahr, zu einer Informations-Monokultur auszuarten, in der letztlich nur noch Google existiert und andere Suchdienste gar nicht mehr wahrgenommen werden.
Nach Meinung des SuMa-eV[4] kann ein Beitrag zur Vermeidung dieses Zustandes sein, alternative Suchdienste zu entwickeln und anzubieten, die in bestimmten Nischen besser bzw. leistungsfähiger als Google sind (vgl. [Mar07] und [SB07]). In der vorliegenden Arbeit soll ein kleiner Beitrag zu so einer Alternative geleistet werden, indem eine qualitativ hochwertige Suchmöglichkeit für den Bereich der Hochschulschriften vorgeschlagen wird.

[1] `http://www.google.de`

[2] `http://de.altavista.com/`

[3] `http://de.yahoo.com/`. Yahoo startete als manuell zusammengestellter Webkatalog, hat inzwischen aber auch eine maschinell zusammengestellte Suchkomponente, die standardmäßig genutzt wird. Der Katalog als Einzeldienstleistung ist nach wie vor unter urlhttp://de.dir.yahoo.com/ zu finden.

[4] `http://suma-ev.de`

Webkataloge

Im Gegensatz zu den in roboterbasierten Suchmaschinen werden Webkataloge nicht durch Robots bzw. Spider zusammengetragen, sondern manuell eingetragen, beschrieben und verschlagwortet (vgl. [Kar03], S. 22). Webkataloge zeichnen sich dadurch aus, dass sie nicht nur mit frei formulierbaren Eingaben durchsucht werden können, sondern der Nutzer auch über einen Kategorienbaum durch das Verzeichnis navigieren kann (vgl. ebd.). Ein bekannter, auf deutsche Seiten begrenzter Webkatalog ist web.de[1]. Auf internationaler Ebene gibt es zum Beispiel Dmoz[2]. Viele andere Kataloge und Suchmaschinen bekommen Suchtreffer auch von dmoz geliefert (vgl. [Kar04]). Unter anderem basiert auch Googles Verzeichnisdienst[3] auf dmoz.
Aufgrund der manuellen Zusammenstellung sind Webkataloge zumeist nicht so umfangreich wie die robotbasierten Suchmaschinen, aus dem gleichen Grund sind sie aber auch genauer und über die Kategrienliste intuitiver durchsuchbar als die maschinell zusammengestellten Engines (vgl. ebd.).

Metasuchmaschinen

Wie in den vorangegangenen Abschnitten beschrieben, gibt es eine Vielzahl von Suchdiensten, mit denen der Nutzer im Internet navigieren und Seiten aus seinem Interessengebiet finden kann. So groß der Index einer einzelnen Suchmaschine auch immer sein mag, so problematisch ist es dennoch, ein vollständiges Suchergebnis zu erzielen, da keine Suchmaschine das gesamte Internet indiziert (vgl. [Kar03], S. 100). So kann mit Hilfe einzelner Suchmaschinen immer nur ein Teilbereich des Gesamtnetzes durchsucht werden. Um dieses Problem zu lösen, gibt es Metasuchmaschinen. Metasuchmaschinen geben eine Suchanfrage an verschiedene Suchmaschinen gleichzeitig weiter und werten die Ergebnislisten der einzelnen Maschinen aus (vgl. ebd., S. 102).

Dirk Lewandowski bestreitet den Nutzen von Metasuchmaschinen:

> Während Meta-Suchmaschinen vor einigen Jahren noch zum Erreichen von Vollständigkeit und zur Erhöhung des Recall als sinnvoll angesehen werden konnten ..., können diese Ziele heute eher von einzelnen algorithmischen Suchmaschinen oder der manuellen Suche in mehreren Suchmaschinen nacheinander erreicht werden. ... Vielmehr dient diese Form der Suchmaschinen mittlerweile eher dem Zweck der Demonstration neuer Suchmaschinen-Technologie. ([Lew05], S. 25)

[1] `http://web.de`
[2] `http://dmoz.org`
[3] `http://www.google.de/dirhp?hl=de`

Der Fakt, keinen eigenen Index halten zu müssen, kann vorteilhaft sein, da weniger Speichervolumen für die Metasuchmaschine notwendig und die Pflege des Index aufwändig ist. Daher räumt auch Lewandowski ein, dass „neue Anbieter ... zunehmend dazu über[gehen], ihre Lösungen als Meta-Suchmaschine aufzusetzen." (ebd.). Für den Endanwender ergibt sich daraus der große Nachteil, dass Metasuche meist deutlich langsamer ist als die Recherche in einer Suchmaschine mit eigenem Index. Daraus, dass verschiedene Fremddatenbanken über deren generische Interfaces abgefragt werden müssen, ergibt sich, dass Metasuchmaschinen bei jeder einzelnen Suchanfrage auf die Suchergebnislisten jedes einzelnen angeschlossenen Suchsystems warten müssen (vgl. [Kar03], S. 102).

Metasuchmaschinen müssen sich insgesamt anderen Problemen stellen als indexbasierte Suchmaschinen. Während bei den indexbasierten Suchern die Ergebnisqualität mit der Qualität des Index steht und fällt, hängt sie bei einer Metasuchmaschine davon ab, wie die Ergebnislisten der Fremdsuchmaschinen verarbeitet werden können. Normalerweise geben die Einzelsuchmaschinen eine HTML (HyperText Markup Language)-formatierte Ergebnisliste zurück. Diese muss von der Metasuchmaschine weiterverarbeitet (geparst) werden. Allerdings kann die Metasuchmaschine dabei nicht feststellen, in welchem Kontext die Suchergebnisse gefunden wurden und welche Metadaten in welcher Reihenfolge in der Ausgabe enthalten sind. Das Parsen und Bewerten der Ergebnisliste ist also das schwierigste Problem beim Aufbau eines Metasuchers. Unter Umständen kann schon eine Änderung im Ausgabeformat das Suchergebnis für die Metasuchmaschine unbrauchbar machen oder verfälschen[1].

Die Metasuchmaschine MetaGer[2] arbeitet nach dem oben beschriebenen Prinzip der Metasuche. MetaGer2[3] geht noch einen Schritt weiter und überprüft die Treffer, die aus der Metasuche gewonnen wurden, noch einmal durch einen direkten Aufruf der gefundenen Seite. Dadurch kann festgestellt werden, ob die Seite überhaupt noch existiert und zudem kann das Relevanzranking verbessert werden, indem überprüft wird, wo und wie auf der Seite die Suchbegriffe vorkommen (vgl. [as05], [FB07b] und [FB07a]).

Dokumentenserver

Unter Dokumentenservern im Sinne der vorliegenden Arbeit werden Server verstanden, auf denen zumeist wissenschaftliche Texte wie Forschungsberichte oder Dissertationen einer breiten Öffentlichkeit zugänglich gemacht werden. Solche Dokumen-

[1] vgl. [SBS98], `http://www.metager.de/inet98/paper.html#3.3`

[2] `http://www.metager.de`

[3] `http://www.metager2.de`

tenserver werden von Universitäten oder Universitätsbibliotheken betrieben, um die von den ansässigen Professoren oder Doktoranden erstellten Arbeitsergebnisse zu verbreiten[1].

Die DINI (Deutsche Initiative für Netzwerkinformation) bietet seit 2004 ein Zertifikat für Dokumentenserver an, mit dem der Betreiber eines solchen Dienstes belegen kann, dass sein Server spezielle Anforderungen, die an so einen Service gestellt werden, erfüllt. Über das DINI-Zertifikat kann nachgewiesen werden, dass der Dokumentenserver gängige Standards befolgt und unterstützt und so in nationale sowie internationale Kooperationen eingebunden werden kann. Die Zertifizierungsbedingungen wurden 2006 überarbeitet, angepasst und erweitert. Daraus entstand das Zertifikat 2007, dessen Voraussetzungen die Serverbetreiber für eine erfolgreiche Zertifizierung aktuell erfüllen müssen[2]. Alle derzeit zertifizierten Server sind noch nach den Bedingungen von 2004 zertifiziert (vgl. [DIN07]).

Zu den Zertifizierungsbedingungen zählt auch, dass der Dokumentenserver OAI-PMH (Open Archives Initiative - Protocol for Metadata Harvesting) unterstützen muss. OAI-PMH ist ein Protokoll, in das verschiedene Metadatenformate gekapselt und so über eine einheitliche Schnittstelle zwischen Diensten ausgetauscht werden können.[3]
Auf OAI-PMH aufsetzende Diensteanbieter werden von der OAI (Open Archives Initiative) in Serviceprovider und Dataprovider unterteilt. Als Dataprovider gelten die einzelnen Dokumentenserver, die ihre Daten über OAI-PMH zur Verfügung stellen. Mindestanforderung an einen Dataprovider ist die Lieferung von Datensätzen im Metadatenformat DC (Dublin Core). Andere Metadatenformate zum Austausch von Daten mit speziellen Diensten können zusätzlich angeboten werden. Die Serviceprovider sind Diensteanbieter, die die von Dataprovidern gelieferten Daten weiterverarbeiten und sie dann wiederum zur Verfügung stellen.[4]
OAI-PMH ist ein reines Harvesting-Protokoll - die Daten werden also von den Serviceprovidern abgeholt (geharvestet). Eine Suchfunktionalität ist für das Protokoll nicht definiert. Serviceprovider können unabhängig von OAI-PMH realisierte Suchfunktionen anbieten. Ein derartiger Serviceprovider ist OAIster, der an der Univer-

[1] Eine Liste aller deutschen Dokumentenserver findet sich unter `http://www.dini.de/dini/wisspub/repositories/german/index.php`.

[2] Die Zertifizierungsbedingungen 2007 sind zu finden unter `http://www.dini.de/fileadmin/docs/dini_zertifikat_2007_v2.1.pdf`

[3] Die Spezifikation von OAI-PMH ist in [OAI04] zu finden

[4] Eine Liste sämtlicher Dataprovider findet sich unter `http://www.openarchives.org/Register/BrowseSites`, eine Serviceproviderliste bietet `http://www.openarchives.org/service/listproviders.html`.

sity of Michigan angeboten wird[1].

Datenbanken

Bibliothekskataloge oder kommerzielle Datenbankanbieter wie DIMDI (Deutsches Institut für Medizinische Dokumentation und Information) sind nicht durch herkömmliche Crawler indizierbar, da sie sich meist hinter Eingabeformularen verbergen, die durch Nutzereingaben ausgefüllt werden müssen. Eine Maschine könnte hier nur automatisiert Begriffe aus Wörterbüchern eingeben, was aber nur in den seltensten Fällen sinnvoll ist. Derartige Dienste können durch die eigenen Eingabeformulare oder durch Metasuchmaschinen (siehe Abschnitt "Metasuchmaschinen" auf Seite 6) abgefragt werden. Für Bibliothekskataloge gibt es eine solche Metasuchmaschine, den Karlsruher Virtuellen Katalog KVK[2].
[Web01] und [Web06] beschreiben drei verschiedene Kooperationsformen zum Einbinden von derartigen Datenbanken in (Meta)-Suchmaschinen:

- Kooperative Datenbankanbieter ermöglichen Suchmaschinenanbietern direkten Zugriff auf die Datenbank.
- Nicht-kooperative Datenbankanbieter ermöglichen Suchmaschinenbtreibern den Zugriff auf die Datenbank nur über das Webformular ohne konkrete Definition von Abfrageschnittstellen und Rückgabeformaten.
- eingeschränkt kooperative Anbieter ermöglichen Suchmaschinenbetreibern den Zugriff auf die Datenbank zwar nur per Webformular, aber mit einem fest formatiertem Rückgabeformat und Bekanntgabe des Anfrageformates.

Der Datenbankbetreiber entscheidet, welche Form der Kooperation verfolgt werden soll und stellt den Suchmaschinenbetreiber darauf basierend die entsprechenden Schnittstellen und Dokumentationen zur Verfügung.

RSS-Feeds

Um Daten zwischen Webservice-Anbietern auszutauschen, werden festgelegte Formate benötigt. Ein Beispiel für ein solches Format ist RSS (Really Simple Syndication). Der Nutzer kann sich mit Hilfe von RSS schnell und einfach über Neuigkeiten oder Änderungen informieren lassen, indem er einen Feed abonniert und ihn in seinem Readerprogramm einträgt.

[1] OAIster harvested per OAI-PMH einzelne Dataprovider und stellt die gesammelten Daten unter einer gemeinsamen Oberfläche zur Verfügung. `http://oaister.umdl.umich.edu/cgi/b/bib/bib-idx?c=oaister;page=simple`

[2] `http://www.ubka.uni-karlsruhe.de/kvk.html`

Technisch enthält RSS nur sehr wenige obligatorische Elemente (vgl. [RSS07]). Gelesen werden kann das Format mit einem Feedreader, von denen es inzwischen eine Vielzahl gibt[1], oder direkt mit dem Browser Mozilla Firefox[2] ab der Version 2.0.

Eine Erweiterung von RSS ist ASF (Atom Syndication Format), das kurz auch einfach Atom genannt wird. Atom enthält mehr Elemente als RSS und geht dabei speziell auf die Bedürfnisse von Weblogs und Newsseiten ein (vgl. [Wik07b]). Beschrieben ist Atom in [Int05].

RSS wird auch im bibliothekarischen Umfeld genutzt. Zum Beispiel bietet die TUB-HH individuell zusammenstellbare RSS-Feeds zu Neuerwerbungen an[3]. Zudem werden die Neuigkeiten der Bibliothek per RSS verbreitet[4] und auch der Dokumentenserver TUBdok unterstützt verschiedene RSS-Feeds.

2.1.2 Suchmaschinentechnik

Aufbau von Suchmaschinen

Nach Eichstädt kann der Vorgang in einer Suchmaschine „vom Einsammeln bis zur Möglichkeit des Durchsuchens der Datenbestände ... in drei Arbeitsschritte unterteilt werden“ ([Eic06][5], S. 19ff). Diese drei Arbeitsschritte sind das Crawlen, das Parsen und das Indizieren. Der Crawler sammelt die Dokumente ein und speichert sie lokal ab. Der Parser verarbeitet das Dokument und zerlegt es in logische Blöcke. Dazu benötigt er reinen Text, das heißt, dass binäre Formate wie PDF (Portable Document Format) zunächst umgewandelt werden müssen. Wie die Umwandlung stattfindet, ist suchmaschinenspezifisch. Einige Suchmaschinen haben dazu spezielle Konverter direkt implementiert, andere greifen auf externe Konverterprogramme wie *pdftotext* zurück. Nach dem Parsen übernimmt der Indexer den Eintrag der einzelnen Dokumentbestandteile in den Datenbestand der Suchmaschine, den sogenannten Index.

Ranking von Suchergebnissen

Aufgrund der großen Menge an indizierten Dokumenten benötigen Suchmaschinen Mechanismen, die Suchtreffer maschinell zu bewerten und dem Nutzer damit An-

[1] Eine Übersicht über Feedreaderprogramme gibt dmoz unter `http://www.dmoz.org/Computers/Software/Internet/Clients/WWW/Feed_Readers/`

[2] `http://www.mozilla-europe.org/de/`

[3] `http://www.tub.tu-harburg.de/mybibrss/`

[4] `http://www.tub.tu-harburg.de/rdf/news-de.rss`

[5] Die Arbeit liegt dem Autoren dieser Arbeit vor, sie ist jedoch nicht veröffentlicht und damit nicht frei verfügbar.

haltspunkte zu geben, wie genau der einzelne Treffer auf seine Anfrage passt. Auf diese Weise kann dem Nutzer trotz hoher Treffermengen ein Hinweis gegeben werden, welche Dokumente er sich am ehesten anschauen kann. Welcher Algorithmus genau verwendet wird, hängt von der Suchmaschine ab.

[Sul07] und [SB98][1] stellen Methoden zum Gewichten von Webseiten vor. Heute bekannte und verwendete Ranking-Algorithmen orientieren sich demzufolge an der Häufigkeit, in der Worte auf der Webseite vorkommen, sowie an der Position, an der diese Worte stehen. Steht der gesuchte Begriff zum Beispiel im Titel der Seite, so bekommt der Treffer eine höhere Gewichtung. Auch eine Position des Wortes am Anfang der Seite kann als Indiz für eine höhere Gewichtung genutzt werden.

Zur Gewichtung werden auch sogenannte "off the page" Ranking-Kriterien verwendet, die der Webmaster nicht so leicht beeinflussen kann. Dazu gehört vor allem die Analyse der Linkstruktur der Seite (vgl. [Sul07]). Der von der Suchmaschine Google verwendete Ranking-Algorithmus Page-Rank basiert vor allem auf einem derartigen Verfahren. Er misst die Popularität einer Seite, indem die Menge der Links bemessen wird, die auf sie verweisen. Auch die anderen bereits genannten Faktoren spielen beim Page-Rank eine Rolle, so zum Beispiel, wie und wo ein Suchbegriff auf der Seite auftaucht (vgl. [Ren07], Folie 5).

Um ein hohes Ranking zu erlangen, versuchen manche Webseitenbetreiber mit fragwürdigen Maßnahmen, die Ranking-Algorithmen der Suchmaschinen auszutricksen. Dieses Vorgehen wird analog zum von eMails bekannten Spam als Spamming bezeichnet. Verbreitet ist z.B. das Spamming per Meta-Tag, bei dem in den Metadaten zu einer Webseite irreführende oder falsche Worte genannt und teilweise übermäßig wiederholt werden (vgl. [SB00b][2]). Auch unlesbare weiße Schrift auf weißem Hintergrund ist ein bekannter Versuch, das Ranking der eigenen Seite zu verbessern (vgl. [SB00b][3]). Um dem Spamming entgegenzuwirken, unternehmen die Suchmaschinen-Betreiber Gegenmaßnahmen. Beispiele dafür sind in [SB00b][4] und in [SB00a][5] genannt.

[1] `http://meta.rrzn.uni-hannover.de/meta-strukt/metager-rank.html` und `http://meta.rrzn.uni-hannover.de/meta-strukt/volltext-rank.html`

[2] `http://www.metager.de/inetbib2000/betr1.html`

[3] `http://www.metager.de/inetbib2000/betr2.html`

[4] `http://www.metager.de/inetbib2000/gegenmass.html`

[5] `http://metager.de/dgd2000/filter.html`

2.1.3 Probleme bei der Suche im Internet

Invisible Web

Unter dem Invisible Web, auch Deep Web genannt, wird die Menge an Internet-Seiten verstanden, die aus verschiedenen Gründen nicht über Suchmaschinen gefunden werden können. Einer dieser Gründe ist zum Beispiel, dass das Format, in dem das Dokument gespeichert ist, nicht von Suchmaschinen indiziert werden kann oder dass der Betreiber der Seite selbige zugriffsgeschützt hat (vgl. [Lew05], S. 51ff[1]).

Vor einigen Jahren gehörten auch PDF-Dokumente in den Bereich des Invisible Web. Diese Einschränkung gilt heute nicht mehr, seitdem sich PDF weitgehend als Dokumentformat durchgesetzt hat und zahlreiche PDF-Ressourcen im Web veröffentlicht sind. Nach wie vor unindizierbar sind jedoch z.B. Inhalte in Flash-Filmen. Problematisch sind auch nicht-textliche Inhalte wie Videos oder Musikstücke. Diese können nicht direkt indiziert werden, sondern nur in Form von Metadaten, die direkt in der Datei oder auf der einbettenden HTML-Seite gespeichert werden.

Datenbanken kommerzieller Anbieter oder Bibliothekskataloge können ebenfalls noch nicht von Suchmaschinen indiziert werden. Um sie mit Suchmaschinen abdecken zu können, muss ein Kooperationsmodell verwendet werden (siehe Abschnitt "Datenbanken" auf Seite 9). Auf diese Weise können mit Hilfe von Metasuchmaschinen auch Teile des Invisible Web erfasst und in der aggregierten Trefferliste ausgeben werden (vgl. [Kar03], S. 152[2]).

Hohe Treffermengen

Heutzutage zählen universitäre Dokumentenserver wie der in dieser Arbeit behandelte TUBdok weitgehend nicht mehr zum Invisible Web und werden aufgrund der freien Verfügbarmachung von Dokumenten, die zumeist im PDF-Format vorliegen, von den großen Suchmaschinen erfasst (vgl. [Lew05], S.66[3]).

Allgemeine Anfragen mit nur einem oder zwei Suchbegriffen ergeben aufgrund der großen Menge indizierter Dokumente eine sehr große Treffermenge, die unüberschaubar ist. Der Nutzer sieht sich daher zumeist nur die ersten Seiten der Suchtreffer an und ist bereits mit einer oberflächlichen Antwort oft schon zufrieden, weil Zeit oder Motivation fehlen, sich auch andere Suchtreffer anzuschauen[4]. Wenn zumeist schon

[1] `http://www.durchdenken.de/lewandowski/web-ir/?35_Das_Invisible_Web.html`.

[2] `http://www.suchfibel.de/5technik/sammeln.htm`

[3] `http://www.durchdenken.de/lewandowski/web-ir/?42_Strukturinformationen_i.html`

[4] Eine Studie der Firma iProspect ergab, dass nur 10% der Nutzer über die dritte Ergebnisseite hinaus klicken (vgl. [iPr06]) und Hendrik Speck berichtet von einer älteren iProspect-Studie, laut

allein eine einzelne Ergebnisseite mit 10 Suchtreffern ausreicht, um das Informationsbedürfnis vermeintlich zu befriedigen, dann werden andere Suchmaschinen zur erweiterten Suche erst recht nicht genutzt. Auf diese Weise kann wissenschaftliches Recherchieren verlernt werden.

Dieses Problem kann nicht nur Teile des Invisible Web (z.B. kommerzielle Datenbankbetreiber) beeinträchtigen, sondern auch nicht-profitorientierte universitäre Einrichtungen und damit Dokumentenserver, die nicht die finanziellen Möglichkeiten haben, durch SEO (Search Engine Optimizing) oder erkaufte Treffer eine gute Position im Ranking der Suchmaschinen zu erlangen. Daher ist es wichtig für diese Gruppen, andere Wege zu finden, sich sichtbar zu machen.

Informationsqualität

Im Prinzip kann jeder Mensch, der des Schreibens mächtig ist und einen Internetzugang hat, Informationen im Internet verbreiten. Vereinfacht wird dies durch das Web 2.0, das entscheidend durch die Beteiligung und Einbeziehung des Nutzers geprägt ist. Beispiele für Web 2.0-Dienste sind Wikis, Weblogs, Foren oder Communities, in denen der Nutzer selbst aktiv werden kann. Das vielleicht bekannteste Projekt aus diesem Bereich ist die Wikipedia[1].

Ein Problem an diesem sehr offenen Zugang zu Wissen ist, dass nicht alle Informationen, die auf diese Weise verbreitet werden, der Wahrheit entsprechen. Es gibt schließlich keine kontrollierende Instanz, die über die Freischaltung von Informationen entscheidet. Im Gegensatz dazu positionieren sich universitäre Dokumentenserver als Werkzeuge, auf denen Content, der von Wissenschaftlern erstellt wurde, bereitgestellt wird. Der Autor ist in diesen Fällen eindeutig erkennbar und identifizierbar und es ist nachweisbar, dass er diesen Content auch wirklich selbst erstellt hat. Die Qualität ist also unzweifelhaft eine andere als bei Wikipedia-Artikeln, die von jedem auch anonym oder unter Vorgebung einer falschen Identität geändert werden können.

der 23% der Nutzer überhaupt nur zur zweiten Ergebnisseite springen (vgl. [ST06], Folie 60).

[1] `http://de.wikipedia.org`

2.2 Dokumentenserver

2.2.1 OPUS

Geschichte

OPUS wurde von der Universitätsbibilothek Stuttgart entwickelt und erstmals auch dort eingesetzt. Die Entwicklung startete im Jahr 1997 und wurde vom DFN (Deutsches Forschungsgsnetz) als Projekt gefördert. Die Software hatte das Ziel, eine Plattform bereitzustellen, auf der elektronische Dokumente gesammelt und verfügbar gemacht werden können. Derartiger Bedarf bestand auch an vielen anderen Hochschulen, daher ist OPUS längst nicht mehr nur auf die Stuttgarter Region begrenzt: „OPUS ist für Hochschulen frei verfügbar und basiert auf ebensolchen Softwarekomponenten. Derzeit wird es an 34 Universitäten ..., an 12 Fachhochschulen ... und an 4 Bibliotheks-Verbünden ... eingesetzt.“ [OPU06]. Derzeit aktuellste Version der Software ist 3.1, die Nachfolgeversion 3.2 ist noch in Vorbereitung und soll noch 2007 erscheinen.

Formate auf OPUS-Systemen

In OPUS kann die Bibliothek, die den Server betreibt, bestimmen, welche Formate zur Abgabe zugelassen sind. Dabei unterscheidet OPUS zwischen Originalformaten und Präsentationsformaten.

Präsentationsformat Das Präsentationsformat ist die Endversion des Dokuments in dem Format, das der Endnutzer lesen soll. Zumeist wird dafür PDF verwendet, in Einzelfällen aber auch HTML. OPUS sieht in der aktuell vorliegenden Version vor, dass zu jedem Eintrag mindestens eine Präsentationsdatei bereitgestellt werden muss.

Originalformat Unter dem Originalformat werden die Dateien verstanden, die der Autor mit seinem Textverarbeitungsprogramm erstellt hat. Dies können z.B. .doc-Dateien aus MS Word oder .odt-Dateien aus OpenOffice 2.0 sein. Der Upload der Originaldateien ist für den Autor optional. Das Originalformat dient der Bibliothek dazu, bei Bedarf durch Konvertierung andere Formate daraus zu erzeugen. Dies kann notwendig sein, wenn alte Formate nicht mehr unterstützt werden oder das originale Präsentationsformat unlesbar geworden ist. Dies ist mit dem Präsentationsformat nicht möglich. Aber auch viele Originalformate sind nicht langzeitarchivierungssicher, daher sollte die Bibliothek das Originalformat am besten sofort in ein archivierungsfähiges Format wie XML (eXtensible Markup Language) konvertieren.

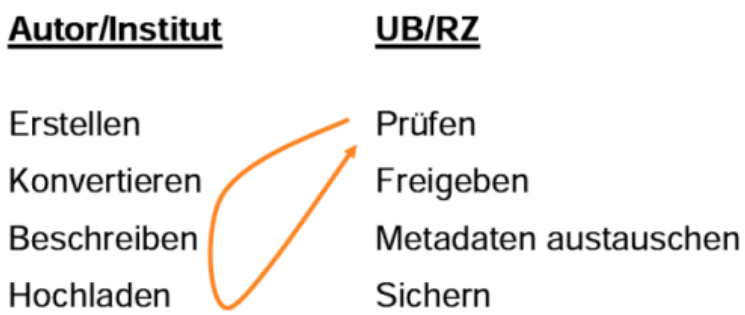

Abbildung 2.3: Workflow zum Publizieren in OPUS ([Sch05], S. 4)

Workflow

Soll eine neue Publikation auf einem OPUS-System eingebracht werden, wird implizit ein Workflow in Gang gesetzt. Dieser ist in Abb. 2.3 dargestellt.

OPUS geht davon aus, dass der Autor seine Arbeit selbst auf dem Dokumentenserver einbringt. Zunächst muss das zu veröffentlichende Dokument fertiggestellt und in ein zugelassenes Präsentationsformat umgewandelt worden sein. Anschließend müssen die Metadaten der Publikation in das Veröffentlichungsformular auf dem OPUS-Server eingetragen werden. Zum Schluss werden alle zur Publikation gehörenden Dateien auf den Server übertragen. Für den Autor ist die Arbeit an dieser Stelle beendet.

Die vom Autor eingetragenen Metadaten landen nach dem vollständigen Ausfüllen des Formulars in einer temporären Tabelle in der OPUS-Datenbank, wo sie nur der Bibliothek zugänglich sind. Die Dateien werden in einem temporären Verzeichnis gespeichert, das in einem zugreifbaren Bereich des Webservers liegt, um die anschließende Bearbeitung durch die Bibliothek zu vereinfachen. Der Zugriff auf das temporäre Verzeichnis ist in der Regel nur Bibliotheksmitarbeitern erlaubt, um mögliche Sicherheitsprobleme zu vermeiden.

Die Bibliothek bekommt nach dem Upload der Datei(en) eine Benachrichtigungs-eMail. OPUS stellt ein internes Administrationsfrontend bereit, in dem dafür zugelassene Bibliotheksmitarbeiter die Metadaten zu jeder Publikation aufrufen können. Der freischaltende Bibliothekar prüft die vom Autor gemachten Angaben auf Plausibilität und formale Korrektheit und stellt sicher, dass sich die eingebrachten Präsentationsformatdateien öffnen lassen. Laufen die Prüfungen positiv, so kann der Eintrag freigeschaltet werden. Bei der Freischaltung werden noch weitere Metadaten hinzugefügt. Der Eintrag wird einer DDC (Dewey-Decimal-Classification)-Sachgruppe zugewiesen und ggf. wird ein URN (Uniform Resource Name) zugeteilt.

Bei der Freischaltung durch die Bibliothek kann auf Knopfdruck eine statische Indexdatei für das Dokument erzeugt werden, auf der die Metadaten angezeigt werden. Zusätzlich zu dieser statischen Seite stellt OPUS auch eine dynamisch aufgebaute Seite zur Verfügung, auf der per Parameter ebenfalls die Metadaten zu jedem auf dem System freigeschalteten Dokument angezeigt werden können. Die statische Seite war in der Anfangszeit von OPUS notwendig, damit die Publikationen ohne Probleme von Suchmaschinen indiziert werden konnte. Heutzutage kommen Suchmaschinen in der Regel auch mit dynamischen Dateien zurecht, so dass die Differenzierung nicht mehr zwingend notwendig ist. Die statische Datei muss bei jeder Änderung an den Metadaten des Dokuments manuell neu aufgebaut werden.

Die letzten beiden in Abbildung 2.3 dargestellten Schritte werden automatisch vorgenommen. Die Dokumente sind nach der Freischaltung über verschiedene Wege zugänglich und können z.B. über die OAI-Schnittstelle geharvested werden. Für die Sicherung der Daten ist die anbietende Institution selbst verantwortlich, OPUS bietet hier keine eingebaute Sicherung.

Schnittstellen

- Web-Frontend mit HTML-Output
  Bis zur Version 3.2 von OPUS wurde HTML als Ausgabeformat erzeugt, ab Version 3.2 ist es XHTML (eXtensible HyperText Markup Language).

- OAI-Schnittstelle zur Übertragung von Daten nach OAI-PMH 2.0
  OAI-PMH ist im Abschnitt "Dokumentenserver" auf Seite 8 detailliert beschrieben.

- Schnittstelle zu den Social Bookmarking-Diensten bzw. Reference Managern Connotea[1] und del.icio.us[2]
  Die Connotea-Schnittstelle wurde vom Autoren dieser Arbeit für die TUB-HH zunächst auf TUBdok entwickelt und in der Version 3.2 in die OPUS-Distribution übernommen. Die Schnittstelle zu del.icio.us wurde von der SULB Saarbrücken entwickelt.

- im Bibliotheksverbund SWB (Süd-West-Verbund) Schnittstelle zur Bibliothekssoftware OCLC PICA zwecks Erleichterung der Katalogisierung

[1] `http://www.connotea.org`
[2] `http://del.icio.us`

Suchfunktionen

Da für die vorliegende Arbeit insbesondere die Suchfunktionalitäten von OPUS von Interesse sind, sollen diese hier kurz betrachtet werden.

Browsingsuche Auf der Webseite des OPUS-Servers kann der Nutzer sich durch verschiedene Listen klicken, um Dokumente von Interesse zu finden. Zum Beispiel lassen sich die von OPUS unterstützten Klassifikationssysteme durchstöbern oder nur bestimmte Dokumenttypen (wie z.B. Dissertationen, Forschungsberichte, etc.). OPUS unterstützt dabei von Haus aus DDC, CCS (Computing Classification System), MSC (Mathematics Subject Classification) und PACS (Physics and Astronomy Classification Scheme). Die Zuordnung eines Dokumenttyps und einer DDC-Klasse ist obligatorisch, die Einordnung in eines der anderen Klassifikationssysteme optional.

Metadatensuche Die Metadatensuche in OPUS geht direkt auf die Datenbank im Backend. Es besteht die Möglichkeit, nur in bestimmten Datenbankfeldern wie *Autor* oder *Publikationsjahr* zu suchen und so seine Suchanfrage einzugrenzen. Realisiert ist die Metadatensuche über eine Eingabemaske, die basierend auf den Eingaben in die Suchmaske und der dazugehörenden Auswahl der jeweiligen Suchfelder eine SQL (Structured Query Language)-Anfrage an die Datenbank formuliert und das Ergebnis dann als HTML ausgibt.

Volltextsuche Bis zur Version 3.0 von OPUS war ht://dig als Suchmaschine in die Software integriert. Ab Version 3.0 wurde die OPUS-eigene Suchmaschine aber abgeschafft und gegen eine Google-basierte Suchmaske ersetzt. Die Google-basierte Suchmaske gibt die eingegebene Suchanfrage einfach an Google weiter und konfiguriert die Suchanfrage automatisch so, dass nur in der eigenen Domäne gesucht wird. Die gewohnte Google-Ergebnisliste wird innerhalb der eigenen Seite dargestellt. Auf diese Weise hat der Anbieter des Dokumentenservers keinen Einfluss auf die Gestaltung des Suchergebnisses und es entsteht eine Abhängigkeit von Google. Alles, was Google nicht indexiert, ist so nicht auffindbar und der Serverbetreiber selbst hat es nicht in der Hand, wann Indexierungsläufe unternommen werden.

Funktionen

Ein Dokumentenserver ist, wie in Abschnitt "Dokumentenserver" auf Seite 7 beschrieben, ein Instrument zum Veröffentlichen elektronischer Publikationen. Daraus ergeben sich zwangsläufig Funktionen, die die zugrundeliegende Software erfüllen

muss. Die Funktionalitäten lassen sich in drei Teile differenzieren: Funktionen für den Serverbetreiber, Funktionen für Autoren und Funktionen für die Leser der Dokumente, also die Nutzer.

Funktionen für den Serverbetreiber

- Unterstützung beim Veröffentlichungsworkflow
  OPUS stellt für die Bibliotheksmitarbeiter eine Administrationsoberfläche zur Verfügung, auf der neu hochgeladene Dokumente freigeschaltet und weitere administrative Aufgaben erledigt werden können.

Funktionen für Autoren

- Unterstützung beim Veröffentlichungsworkflow
  Der Autor wird über Formulare und Hilfetexte durch die Publikation geleitet.

- Lizenzauswahl für die Publikationen
  OPUS bietet dem Autor seit Version 3.0 vom Anbieter einstellbare Lizenzmodelle für seine Publikation an. In der Lizenz wird festgelegt, was der Endnutzer/Leser unter welchen Bedingungen mit der Publikation tun darf. Voreingestellt in OPUS sind an dieser Stelle zwei proprietäre Lizenzmodelle. Diese gestatten es dem Leser, die Publikation zu lesen und im Rahmen der geltenden Regelungen zu zitieren, der Serverbetreiber darf bei einem der Modelle Print-on-Demand für das Dokument anbieten, bei dem anderen Modell ist Print-on-Demand nicht erlaubt. Zusätzlich werden einige CC (Creative Commons)-Lizenzmodelle[1] unterstützt.

Funktionen für Nutzer/Leser

- Bereitstellung von Suchfunktionalitäten
  Die von OPUS gebotenen Suchfunktionalitäten wurden im Abschnitt "Suchfunktionen" auf Seite 17 detailliert beschrieben.

- Print-on-Demand-Service
  Ist Print-on-Demand vom Autoren zugelassen und hat der Serveranbieter entsprechende Partner, die diesen Service realisieren, so hat der Leser die Möglichkeit, Dokumente als gebundenen Ausdruck zu bestellen.

- RSS-Feed für die neuesten Publikationen

[1] Der Autor selbst kann in der Lizenz genau spezifizieren, unter welchen Bedingungen seine Arbeit nutzbar sein soll. Eine Übersicht über vorhandene CC-Lizenzen oder CC-ähnliche Lizenzen sowie ein Baukasten zum Zusammenstellen der eigenen Lizenz gibt es unter `http://creativecommons.org/license/`.

2.2.2 TUBdok

Geschichte

Seit 2002 setzt auch die TUBHH das OPUS-System als Dokumentenserver ein. 2005 wurde der OPUS-Server von Seiten der TUBHH beträchtlich überarbeitet und erhielt ein komplett neues Outfit sowie diverse neue Funktionen. Die Entwicklungen wurden vom Autor dieser Arbeit vorgenommen und sind in [Mar05] öffentlich dokumentiert. Nach Abschluss der Arbeiten erhielt das System den Namen TUBdok. Die Anzahl der Publikationen auf TUBdok hat sich seit diesem Zeitpunkt in etwa verdoppelt.
TUBdok ist seit Anfang 2006 bei der DINI als standardkonformer Dokumentenserver nach den Zertifikatsrichtlinien 2004 zertifiziert (siehe Abschnitt "Dokumentenserver" auf Seite 8).

Ziele

TUBdok unterstützt wie viele andere Dokumentenserver die Publikation nach dem OA (OpenAccess)-Prinzip[1].

Unterschiede zum Originalsystem

- RSS-Feeds auch für Neueinträge in einzelnen Fachgebieten
- Publikationsformular auf verschiedene Seiten aufgeteilt
- Schaffung eines persönlichen Bereiches für Autoren
  Im persönlichen Bereich des Autoren lassen sich Statistiken zum Aufruf der eigenen Dokumente einsehen, persönliche Bookmarks pflegen, und der Autor erhält eine Übersicht über die eigenen Publikationen.
- Integritätsprüfung für die Dokumente anhand von Prüfsummen und Digitalen Signaturen mit GPG (Gnu Privacy Guard)
  Ab Version 3.2 ist diese Erweiterung auch in der offiziellen OPUS-Distribution enthalten.
- datenbankbasiertes Hilfesystem statt statischer Hilfeseiten
- Login zum Zugang zum Publikationsformular
  Im OPUS-Original findet die Autorisierung zum Upload von Daten über das Publikationsformular entweder per IP-basierte Einschränkung statt oder ist

[1] http://www.tub.tu-harburg.de/3910.html

gänzlich offen. Ein Authentifizierungssystem ist nicht Bestandteil von OPUS. Für TUBdok wurde eine Authentifizierung über die Zugangsdaten des Bibliotheksausweises realisiert. Die Abprüfung des Passwortes läuft gegen das lokale Bibliothekssystem.

Suchfunktion

Ergänzend zur Browsingsuche, die in Abschnitt "Browsingsuche" auf Seite 17 beschrieben wurde, unterstützt TUBdok die Klassifikation nach GBV-Basisklassifikation.
Als Suchmaschine für TUBdok wird seit 2005 Mnogosearch[1] genutzt.

[1] `http://search.mnogo.ru`

Kapitel 3

Suchsysteme

3.1 Voraussetzungen für das Suchsystem

Es gibt bereits zahlreiche Software-Bibliotheken, die Strukturen und API (Application Programming Interface)s für die Implementierung von Suchmaschinen zur Verfügung stellen. Für die Realisierung des Suchsystems in dieser Arbeit reicht es aus, eine solche Software-Bibliothek auszuwählen und diese in der Eigenentwicklung zu nutzen.

Um das zu diesem Zweck passendste Produkt auszuwählen, wurden zunächst Anforderungen definiert, auf die einzelne Systeme überprüft wurden. Die Anforderungen ergaben sich aus dem gewünschten Zielsystem und der Umgebung, also dem Dokumentenserver OPUS.

Anforderungen an die Software-Bibliotheken

- OpenSource
  OpenSource-Software ist kostenfrei und leichter an die eigenen Bedürfnisse anpassbar, da direkte Einblicke und Modifikationen im Quellcode möglich sind.
- lauffähig unter Linux
  OPUS ist zum Betrieb auf Linux-Plattformen ausgelegt, daher sollte die Suchsoftware auch auf dieser Plattform betrieben werden können.
- möglichst niedrige Installationsanforderungen
  Wird das Suchsystem später in die OPUS-Distribution übernommen, so ist eine einfache Installierbarkeit wünschenswert, damit alle OPUS-Betreiber die Suchsoftware ohne großen Aufwand übernehmen können.
- bevorzugte Programmiersprache PHP (PHP Hypertext Preprocessor) oder API für PHP vorhanden

Da OPUS in PHP geschrieben ist, ist die Integration eines anderen Systems am einfachsten und effizientesten möglich, wenn es entweder selbst auch in PHP geschrieben ist oder eine API zur Verfügung stellt, die eine unmittelbare Einbindung ermöglicht.

3.2 Prüfung von Suchmaschinen-Bibliotheken

In diesem Kapitel soll überprüft werden, inwiefern einzelne Suchmaschinenprodukte die in Abschnitt 3.1 definierten Anforderungen erfüllen. Auf Grundlage dieser Ergebnisse wird eine Entscheidung für das in dieser Arbeit einzusetzende Suchsystem getroffen.

Im Abschnitt "Aufbau von Suchmaschinen" auf Seite 10 wurden die drei technischen Komponenten einer Suchmaschine vorgestellt. Die im Suchsystem zu indizierenden Daten lagen im Fall dieser Arbeit bereits lokal vor, daher waren nur Parser und Indexer von Bedeutung. Das Crawlen eines bekannten Verzeichnisbaumes auf der lokalen Festplatte ist ausreichend trivial. Daher wurden die Crawlereinheiten der Suchmaschinen-Bibliotheken nur insoweit betrachtet, ob sie generell die Indizierung lokaler Datenbestände ermöglichen.

Softwarebetrachtung Die in diesem Kapitel vorgestellten Softwareprodukte wurden zunächst oberflächlich aufgrund ihrer Beschreibung auf eine theoretische Eignung im Sinne der vorliegenden Arbeit überprüft. Kam die Software nach dieser oberflächlichen Prüfung in Frage, wurde versucht, sie auf einem Testsystem zu installieren. Nach einer erfolgreichen Installation sollte mit Hilfe des jeweiligen Indexers der Datenbestand von TUBdok indiziert werden. Anschließend wurde mittels einer Testrecherche die Qualität des Index geprüft. Nicht erfolgreich auf dem Testsystem installierbare Software fiel direkt aus der Entscheidungswahl.

Indizierungstest Für die erprobten Systeme wurde ein Index über den TUBdok-Datenbestand an Volltexten erstellt. Der TUBdok-Datenbestand besteht größtenteils aus PDF-Dokumenten, enthält aber auch einige PS (PostScript)- und HTML-Dokumente. Zusätzlich liegen einige Originalformate (z.B. Microsoft Word) und Signaturdateien vor.

Indizierungsverhalten Bei der Indizierung wurde betrachtet:

- Wie ist der Indexer zu handhaben?

- Wie geht der Parser bzw. Indexer mit Fehlern um, die zum Beispiel bei der Indizierung fehlerhafter PDF-Dokumente auftreten können?
- Wie kann ein Update des Index vorgenommen werden bzw. ist ein Update überhaupt möglich?
- Wie können neue Dokumenttypen in den Indexer eingebunden werden bzw. ist die überhaupt möglich?

Testrecherche Nach Fertigstellung des Index wurde mittels Testrecherchen untersucht, wie seine Qualität ist bzw. wie vollständig er ist und wie das Grundsystem mit Suchanfragen umgeht. Bei dieser Testrecherche wurde ein besonderes Augenmerk darauf gelegt, inwiefern Operatoren unterstützt werden und welche Suchmöglichkeiten sich ohne zusätzliche eigene Implementierungsarbeit nutzen lassen.
Als Test-Suchanfrage wurde "Bernhard Sell" eingegeben. Auf TUBdok ist ein Dokument publiziert, von dem Bernhard Sell der Autor ist. In anderen Dokumenten wird er refernziert.
Jedes Dokument hat in OPUS eine eindeutige ID. Als optimales Suchergebnis für die Testanfrage konnten folgende IDs erwartet werden:

1. Bernhard AND Sell: 38, 60, 64, 65, 333
2. Bernhard OR Sell: 38, 60, 64, 65, 333, 252, 164, 155, 102, 330, 110, 76
3. Bernhard XOR Sell: 252, 164, 155, 102, 330, 110, 76
4. Bernhard NOT Sell: 76
5. Sell NOT Bernhard: 252, 164, 155, 102, 330, 110
6. Bernhard AND *Sell*: ID 37, 91, 68, 228, 320, 59, 182, 260, 98
7. "Bernhard Sell" (der Suchterm als Phrase): 38, 60

Am tatsächlichen Suchergebnis konnte abgelesen werden, welche Art von Verknüpfung die Suchmaschine automatisch verwendet, wenn nicht explizit ein Operator angegeben wird.

Integrierbarkeit War das Ergebnis der Testrecherche zufriedenstellend, wurde geprüft, inwieweit das System beeinflusst werden kann und wie es in TUBdok bzw. OPUS integrierbar ist. Hierzu wurde geprüft, welche Möglichkeiten das Suchsystem zur Einbindung in PHP bietet.

Optimal an dieser Stelle wäre eine API für PHP, um das Suchsystem in die Struktur des übrigen OPUS-Frontends einbinden zu können. Möglich wäre aber auch eine Abfrage des Suchsystems über eine Ausführung externer Kommandos und anschließendes Parsen des Ergebnisses in PHP. Hierbei wäre auch eine template-basierte Ausgabe oder eine Parametrisierung des Ausgabeformates hilfreich, damit das Suchergebnis ohne großen Zusatzaufwand weiterverarbeitet werden kann.

3.2.1 Terrier

Über die Software Terrier[1] ist eine Plattform zur Entwicklung von Information-Retrieval-Applikationen und wurde an der Universität Glasgow entwickelt. Die Software ist in Java geschrieben und beschränkt sich auf die Bereitstellung von Funktionalitäten zum Parsen und Indizieren von Inhalten. Eine Crawlereinheit ist nicht inbegriffen.[2]

Entwicklungsaktivität Die aktuell letzte Version von Terrier ist 1.0.2 und auf März 2005 datiert (Stand: Juni 2007). So entsteht der Eindruck, dass die Weiterentwicklung gestoppt wurde. In den Foren auf der Seite herrscht noch eine leichte Aktivität.

PHP-Bindung Eine Bindung von Terrier an PHP ist nicht verfügbar.

Installation ÿÿ Für den Download von Terrier ist eine Registrierung erforderlich, nach der eine eMail mit den Daten zum Downloaden versendet wird. Die Software muss nicht kompiliert werden, die Java-Binaries liegen bereits vorkompiliert vor. Ein einfaches Entpacken des heruntergeladenen Archivs reicht aus. Eine Anleitung zum Ausprobieren der Software findet sich in [Ter05].

Indexerzeugung Zunächst wurde mit dem Kommando *trec_setup.sh* die Datenbank für den Index vorbereitet. Anschließend wurde mit Hilfe des Kommandos *find* eine Liste der zu indizierenden Dateien vorbereitet und in der Datei *collection.spec* gespeichert. Die Signaturdateien wurden nicht in die Liste aufgenommen.

Indizierungslauf Nach den Vorbereitungen, die wie in [Ter05] beschrieben durchgeführt wurden, konnte mittels des Scripts *trec_terrier.sh* der Indizierungslauf gestartet werden. Dieser wurde mehrfach von unerwarteten Java-Fehlern unterbrochen.

[1] Terabyte Retriever, `http://ir.dcs.gla.ac.uk/terrier/`

[2] Ein auf Terrier zugeschnittener Crawler ist die gleichfalls an der Universität Glasgow entwickelte Software Labrador (`http://www.dcs.gla.ac.uk/%E7craigm/labrador/`).

Die Dateien, bei denen der Fehler auftrat, wurden daraufhin aus der Indizierungsliste entfernt, bis der Indizierungslauf erfolgreich abgeschlossen werden konnte.

Testrecherche Die oben beschriebene Testrecherche wurde analog zur in [Ter05] beschriebenen Vorgehensweise durchgeführt. Jedoch führte sie zu keinem Erfolg, weil eine topic-Datei nicht gefunden wurde. Da nicht zu ermitteln war, was diese fehlende Datei enthalten soll und nach welchem Schema sie aufgebaut sein soll, wurden die Tests mit Terrier an dieser Stelle abgebrochen.

Fazit Terrier scheint am ehesten von akademischem Interesse zu sein und könnte bei der Entwicklung ganz neuer Erschließungsmodelle genutzt werden, da von Haus aus mehrere Retrievalmodelle unterstützt werden. Es existieren viele wissenschaftliche Untersuchungen und Papiere im Kontext von Terrier, in Produktionssystemen scheint die Engine aber nur sehr selten eingesetzt zu werden. Da in dieser Arbeit eine Suchmaschine nicht nur modelliert, sondern auch realisiert werden soll, und das akademische Interesse eher auf Entwurfsmuster ausgerichtet ist, kommt Terrier im vorliegenden Fall nicht zum Einsatz.

3.2.2 ASPSeek

Über die Software Die Software ASPSeek (`http://www.aspseek.org/`) ist in C++ programmiert und arbeitet per CGI (Common Gateway Interface)-Schnittstelle. Das System benötigt eine MySQL-Datenbank als Backend.

Entwicklungsaktivität ASPSeek scheint nicht mehr weiterentwickelt zu werden. Einem Hinweis auf der Webseite des Produktes ist zu entnehmen, dass 2003 die letzten Änderungen vorgenommen wurden. Die letzten Binärpakete der Software sind vom 21. Oktober 2002, die Sourcen liegen zuletzt von der Version 1.2.10 vor und datieren im tar.gz-Archiv auf den 22. Juli 2002 (Stand: Juni 2007).

PHP-Bindung Eine direkte Einbindung in PHP ist nicht verfügbar.

Installation Zunächst wurde versucht, ASPSeek per Kompilierung der Sourcen für die aktuellste Version auszutesten. Beim Kompilieren mit make traten jedoch Fehler auf, die vermutlich an der eingesetzten Compilerversion (gcc 4.0.1) lagen (vgl. [Neb04][1]). [Neb04] beschreibt die Kompilierung der Entwicklerversion von ASPSeek mit moderneren Compilern. Dieser Versuch scheiterte jedoch beim CVS-Login, das

[1] `http://www.nebel.de/projekte/Vortrag-20041122/Installation(2).html`

nach einem Timeout abgebrochen wurde. Dies machte es unmöglich, die Entwicklerversion zu beziehen. Auch die vorhandenen Binärpakete konnten aufgrund von Kompatibitätsschwierigkeiten nicht installiert werden.
Aufgrund dieser massiven Installationsschwierigkeiten wurde die Betrachtung von ASPSeek an dieser Stelle abgebrochen.

Fazit Michael Nebel sieht einigen fundamentalen Erweiterungsbedarf für die Software (vgl. [Neb04][1]). Dieser wurde bislang nicht umgesetzt. Aus diesem Grund sowie wegen der nicht erfolgreichen Installation scheidet ASPSeek als Grundlage für die Eigenentwicklung in dieser Arbeit aus.

3.2.3 ht://dig

Über die Software Die Software ht://dig[2] ist in C++ programmiert und arbeitet per CGI-Schnittstelle.

Entwicklungsaktivität Die Software wird offensichtlich nicht mehr weiterentwickelt, die letzte Version 3.2.0b6 datiert auf Juni 2004 (Stand: Juni 2007).

PHP-Bindung Eine direkte Einbindung in PHP ist nicht verfügbar.

Installation Zum Kompilieren von ht://dig auf dem Testsystem mussten zwei Zeilen in der Quelldatei htsearch/Collection.h modifiziert werden, wie in [Col07] beschrieben. Danach lief die Kompilierung problemlos durch.

Indexerzeugung Zum Indizieren wurde das von der Software bereitgestellte Programm *htdig* benutzt, das per Kommandozeile aufgerufen wird und per config-Datei gesteuert werden kann. Das Programm umfasst auch einen Crawler. Es muss ein Start-URL (Uniform Resource Locator) angegeben werden. Zu Testzwecken wurde hier zunächst ein lokaler URL über das file-Protokoll angegeben (`file:///usr/local/volltexte`).

ASCII-Index htdig hat auf der Kommandozeile verschiedene Optionen. So ist es z.B. möglich, eine ASCII-Version des Indexes zu erstellen, die auch mit anderen Programmen einfach durchsucht werden kann, so dass zum Indizieren *htdig*, aber zum Präsentieren der Suchergebnisse auch andere Programme benutzt oder ganz neue Suchmechanismen darüber aufgesetzt werden können.

[1] `http://www.nebel.de/projekte/Vortrag-20041122/Weiterentwicklung(2).html`
[2] `http://www.htdig.org/`

Parser ht://dig hat einen integrierten PDF-Parser, der auf der externen Software *acroread* von der Firma Adobe aufsetzt (vgl. [ht:05]). Dieser interne Parser kann aber auch durch externe ersetzt werden. Bei ht://dig mitgeliefert sind Converterscripte für verschiedene Dateitypen wie PDF oder doc enthalten, die ihrerseits auf Fremdsoftware zurückgreifen. Standardmäßig wird dabei für PDF *pdftotext* bzw. für doc *catdoc*[1] verwendet. Die gewünschten Converterprogramme können über die Konfigurationsoption external_parser (vgl. [htd02][2]) in die *htdig.conf* eingebunden werden (vgl. [htd05]). Der Testindex wurde unter Zuhilfenahme des bei ht://dig mitglieferten Perl-Programmes *doc2html.pl* erstellt.

Testrecherche Die testweise Suche wurde über das von ht://dig bereitgestellte CGI-Programm *htsearch* abgewickelt, das über den Webbrowser aufgerufen wurde. Die Recherche nach "Bernhard Sell" ergibt 240 Treffer und damit wesentlich mehr als erwartet, selbst wenn eine Standard-Verknüpfung per OR angenommen wird. Das PDF von Herrn Sell ist nicht darunter, nur die dazugehörige, von OPUS erzeugte, statische Indexseite (siehe Abschnitt 2.2.1).

Einbindung in OPUS Wie in Abschnitt 2.2.1 angesprochen, war ht://dig bis zur Version 3.0 von OPUS als Standard-Volltextsuchmaschine integriert. Da eine direkte Einbindung in PHP nicht möglich war, wurde ht://dig als externe CGI-Anwendung gestartet und erzeugte als solche auch die HTML-Ausgabe. Daher brachte diese Einbindung einen enormen zusätzlichen Pflegeaufwand mit sich. Allein eine Anpassung des Layouts von OPUS war problematisch, weil für die ht://dig-Oberfläche diese Änderungen separat nachgepflegt werden mussten.

Fazit Die Qualität des Index ist nicht ausreichend, wie die zu hohe Anzahl irrelevanter Treffer bei der Testrecherche zeigt. Daher scheidet ht://dig für diese Arbeit aus.

3.2.4 mnoGoSearch

Über die Software mnoGoSearch[3] ist in C++ geschrieben und stützt sich auf eine MySQL-Datenbank. Die Software lässt sich per Kommandozeile bedienen, es gibt aber auch verschiedene grafische Frontends. Unter Linux/Unix ist die Software kostenfrei, unter Windows wird eine Gebühr verlangt.

[1] `http://www.wagner.pp.ru/~vitus/software/catdoc/`
[2] `http://www.htdig.org/attrs.html#external_parsers`
[3] `http://search.mnogo.ru/`

Entwicklungsaktivität mnoGoSearch wird kontinuierlich weiterentwickelt, die aktuellste Version für Linux/Unix ist derzeit 3.3.3 vom 08.05.2007, die aktuellste Version unter Windows ist noch 3.2.42 (Stand: Juni 2007).

PHP-Bindung mnoGoSearch ist komplett in PHP integrierbar, entweder beim Kompilieren von PHP oder später als Extension. Über das Paket wird die Nutzung zusätzlicher PHP-Funktionen über eine eigene API ermöglicht. Auch eine beispielhafte Umsetzung als volle Suchmaschine unter PHP ist erhältlich.

Installation Wie in Abschnitt 2.2.2 kurz erwähnt, wurde mnoGoSearch im Rahmen der Umstellung des Layouts im Jahr 2005 in TUBdok integriert, somit existierten dazu bereits einige Erfahrungswerte und eine Testinstallation lag sogar bereits im OPUS-Kontext vor. Auf TUBdok und dem TUBdok-Testsystem ist mnoGoSearch direkt in PHP einkompiliert und funktioniert bislang ohne Probleme.

Indexerzeugung Die Indexerzeugung wird per Kommandozeilentool gestartet, das *indexer* heißt. Im Fall von TUBdok wird täglich ein Indizierungslauf unternommen, der die komplette bisherige Indexdatenbank verwirft und neu aufbaut. Dieses Prinzip ist im Allgemeinen zwar ressourcenintensiver als ein Update, jedoch entfällt dafür die Arbeit, zu überprüfen, was sich geändert hat. Da der Datenbestand von TUBdok mit etwa 300 Dokumenten nicht besonders groß ist, ist diese Vorgehensweise akzeptabel, für größere OPUS-Instanzen wäre er es jedoch nicht.

Die maximale zu indizierende Dateigröße kann in der Konfigurationsdatei des Indexers *indexer.conf* eingestellt werden. Ebenfalls können in dieser Datei bestimmte reguläre Ausdrücke in URLs vom Indizierungsvorgang ausgeschlossen werden. Auf TUBdok werden auf diese Weise die Signaturdateien mit der Endung .asc oder .sig ausgeschlossen sowie die Dateien des Originalformates, falls solche vorhanden sind. Externe Parser zur Verarbeitung beliebiger Fremdformate lassen sich auch über *indexer.conf* einbinden. Für PDF-Dokumente wird auf TUBdok der *pdftotext*-Parser verwendet. mnoGoSearch kann lokale Pfade indizieren und damit den ansonsten notwendigen HTTP (HyperText Transfer Protocol)-Traffic einsparen. Dabei kann der lokale Pfad im Index durch eine HTTP-Adresse ersetzt werden, damit die Suchergebnisse in der Ausgabe über den Browser korrekt verlinkbar sind.

Testrecherche Die Suche nach "Bernhard Sell" ergibt bei der installierten Version 3.2.35 auf dem TUBdok-Produktivsystem gar keine Treffer. Auf dem Testsystem sind es zwei Treffer. Beim ersten der beiden handelt es sich um die statische Indexdatei der Arbeit von Herrn Sell, der zweite Treffer ist ein PDF, in dem die Arbeit

referenziert wird. Das sind zu wenige Treffer. Wird in der TUBdok-Suchoberfläche[1] die Option "enthalten in" statt der Suche nach "dem ganzen Wort" gewählt, so ergeben sich 4 Treffer (ID 60, 68, 228, 320). Auch dieses Suchergebnis entspricht nicht den Erwartungen und ist nicht vollständig.

Fazit mnoGoSearch bietet aufgrund der sehr guten Integrierbarkeit in PHP viele Möglichkeiten zur Einbindung in OPUS/TUBdok. Ärgerlich sind jedoch die Ungenauigkeiten in der Indexbildung bzw. bei der Suche. Eine Analyse des Indexlaufes ergab keine ergiebigen Hinweise auf die Ursachen der Defizite. Sollte mnoGoSearch weiterverwendet werden, müsste dieses Problem nochmal genauer analysiert werden.

3.2.5 Perlfect Search

Über die Software Perlfect Search[2] ist eine Sammlung von Perl-Scripts, die zusammengenommen als Grundlage einer Suchmaschine dienen können. Umgesetzt sind dabei eine Indizierungseinheit und eine Benutzeroberfläche zur Suche.

Entwicklungsaktivität Die Entwicklung scheint noch aktiv weitergeführt zu werden, so datiert die letzte Version 3.37 auf den 30. März 2007 (Stand: Juni 2007).

PHP-Bindung Eine Umsetzung oder Anbindung in PHP ist nicht verfügbar.

Installation Die Installation lief ohne Schwierigkeiten über ein Perl-Installationsskript, über das abfragebasiert bestimmte Parameter wie der Pfad zum cgi-bin-Verzeichnis des Webservers oder der Pfad zum Webserver-Hauptverzeichnis angegeben werden können.

Indexerzeugung Der Indizierungslauf kann per Kommandozeile oder über den Webbrowser gestartet werden. Zum Testen wurde nur die kommandozeilenbasierte Indizierung durchgeführt. Bei der Indizierung können zwei verschiedene Methoden eingestellt werden: über das lokale Dateisystem oder per HTTP vom Webserver. HTTP ist dann angeraten, wenn dynamische Dateien (z.B. PHP-basierte Seiten) indiziert werden sollen. In solchen Fällen würde bei der Indizierung über das Dateisystem der Quelltext indiziert, und nicht der Output. Da im Fall OPUS/TUBdok keine dynamischen Dateien indiziert werden sollen, wird über das lokale Dateisystem indiziert.

[1] `http://doku.b.tu-harburg.de/mnogosearch/search.php?la=de`
[2] `http://perlfect.com/freescripts/search/`

Konfiguration In einer Configdatei können zu indizierende Dateiendungen angegeben und externe Parser zur Verarbeitung von Fremdformaten eingebunden werden. Standardmäßig wird *pdftotext* für PDF-Dokumente verwendet.

Umgang mit Fehlern Der Indexer meldet eventuelle Fehler in den PDFs, geht aber ohne Abbruch darüber hinweg.

Testrecherche Die Testrecherche ist wenig befriedigend. Es werden bei einer Anfrage nach "Bernhard Sell" 260 Dokumente ausgegeben. Diese Anzahl entspricht in keiner Weise den oben definierten Erwartungen. Auch eine fehlerhafte UND-Verknüpfung kann ausgeschlossen werden, da die Anzahl der Treffer auch bei einer ODER-Suche nicht stimmt. Positiv fällt jedoch auf, dass das PDF von Herrn Sell an erster Stelle auftaucht und damit offenbar ordnungsgemäß indiziert wurde.

Fazit Aufgrund der vielen irrelevanten Treffer und der nicht vorhandenen direkten PHP-Schnittstelle wird PerlfectSearch für diese Arbeit nicht weiter betrachtet.

3.2.6 Namazu

Über die Software Namazu[1] ist eine unter GPL (GNU General Public Licence)-Lizenz stehende Volltextindizierungs-Software, die ein vorgefertigtes Suchsystem per Kommandozeile oder CGI-Webinterface zur Verfügung stellt. Namazu kommt ursprünglich aus Japan, so ist sie primär auf japanischen Zeichensatz ausgelegt. Inzwischen beherrscht sie aber auch englischen Zeichensatz. ISO-8859-* Zeichensätze werden bislang nicht offiziell unterstützt.

Entwicklungsaktivität Die Entwicklung scheint noch aktiv voranzugehen; die letzte verfügbare Version 2.0.17 datiert auf den 12. März 2007 (Stand: Juni 2007).

PHP-Bindung Eine Umsetzung oder Anbindung in PHP ist nicht verfügbar.

Installation Die Kompilierung und Installation von Namazu funktionierte auf dem Testsystem ohne Probleme.

Indexerzeugung Zum Aufbau des Index stellt Namazu das Kommando *mknmz* zur Verfügung. Der Crawler läuft über die lokale Festplatte und indiziert dabei alles, was im angegebenen Verzeichnis und den Unterverzeichnissen liegt und indizierbar ist. Um Namazu später von einer Weboberfläche nutzen zu können und bei den

[1] `http://www.namazu.org/`

Suchtreffern Hyperlinks statt lokaler Dateireferenzen anzeigen zu lassen, kann eine –replace-Option an *mknmz* übergeben werden, über die per regulären Ausdruck der Dateisystempfad in einen URL umgewandelt werden kann.

Konfiguration Die maximal zu indizierende Dateigröße kann in der config-Datei für den Namazu-Indexer festgelegt werden und wurde für den vorliegenden Test auf 40 MB eingestellt. Der Index muss nicht jedesmal von Neuem aufgebaut werden, Namazu prüft bei einem erneuten Indizierungslauf, welche Dateien eingefügt werden müssen und aktualisiert den Index automatisch. Eine Erweiterung der Software auf neue Dateitypen mit Hilfe externer Parser- oder Converterprogramme ist offenbar nicht vorgesehen; entsprechende Optionen waren in der Configdatei nicht zu finden. Allerdings werden die gebräuchlichsten Formate von Haus aus sehr zufriedenstellend unterstützt.

Testrecherche Die Recherche nach "Bernhard Sell" bringt die erwarteten sechs Treffer: das PDF von Herrn Sell steht an erster Stelle, danach folgt die Indexseite, bei der er in den Metadaten auftaucht. Fälschlicherweise wird allerdings auch Thomas Hapke als weiterer Autor im Suchergebnis angegeben, und zwar gleich doppelt. Bei der Ursachenforschung wurde festgestellt, dass die von OPUS erstellte Indexseite in den <META>-Tags tatsächlich Thomas Hapke als "author" nennt. Die Ergebnisse dieser Testrecherche sind also sehr zufriedenstellend.

Integrierbarkeit Namazu bietet die Möglichkeit, verschiedene Templates zu nutzen und so die Formatierung der Ergebnisliste zu bestimmen. Über die Configdatei *namazurc* kann ein Verzeichnis bestimmt werden, in dem die Templates liegen. Zum Testen wurde ein neues Template angelegt, in dem die Treffer in ein minimales XML-Format überführt wurden. Bei der Suche über Kommandozeile kann per Parameterübergabe ein Rückgabeformat ausgewählt werden. Die Testrecherche wurde über *namazu -hs "Bernhard Sell"/tmp/namazu_index* ausgeführt. Über die Option h wird bestimmt, dass der Output templatebasiert ausgegeben werden soll, die Option s wählt die Short-Anzeige der Treffer aus, für die die neue Templatedatei definiert wurde. Wie erwartet gibt der Befehl eine XML-formatierte Ergebnisliste zurück.

Fazit Namazu macht einen sehr guten Eindruck, was die Qualität des Suchergebnisses angeht. Was fehlt, ist aber eine direkte PHP-API, so dass Namazu nur über die Ausführung auf Systemebene oder über das CGI integriert werden könnte.

3.2.7 WebGlimpse

Über die Software Das amerikanische WebGlimpse[1] ist für die persönliche Nutzung sowie für Non-Profit-Organisationen kostenfrei einsetzbar. WebGlimpse ist eine CGI-basierte Suchoberfläche, darunter liegt die Software Glimpse, die zur Indizierung lokaler Datenbestände dient.

Entwicklungsaktivität Die Entwicklung wird offensichtlich fortgesetzt, die aktuellste Version von WebGlimpse ist 2.17.3 vom 22.03.2007 (Stand: Juni 2007).

PHP-Bindung Eine Umsetzung in PHP ist nicht verfügbar.

Installation Zunächst muss Glimpse als Grundlage von WebGlimpse kompiliert und installiert werden. Die Installation von Glimpse und WebGlimpse funktionierte auf dem Testsystem ohne Probleme.

Indexerzeugung Mit dem Kommando *wgcmd* kann per Kommandozeile ein neues "Archiv" erstellt werden. Dabei wird der Benutzer menugesteuert durch die Erstellung geleitet. Standardmäßig werden nur HTML-Dateien indiziert, für PDF-Unterstützung muss umkonfiguriert werden. Zum Updaten des Index reicht es aus, im Indexverzeichnis das Kommando *wgreindex* auszuführen.

Testrecherche Die Testrecherche nach "Bernhard Sell" wird über das menugesteurte Programm *wgcmd* durchgeführt. Sie ergibt drei Treffer, wobei das PDF zur ID 38 doppelt angezeigt wird. Die Ergebnisanzeige ist sehr unübersichtlich. Es scheint, dass das Programm nach dem Suchbegriff als Phrase gesucht hat.

Integrierbarkeit Die Integrierbarkeit stellt ein großes Problem dar. Da die Kommandozeilenprogramme von WebGlimpse menugesteuert sind, ist hier eine Automatisierung kompliziert, wenn sie überhaupt möglich ist. Es ist nicht ersichtlich, wie eine Übergabe von Parametern aus Fremdprogrammen an das Menu möglich sein kann, was aber für die Integration unbedingt erforderlich wäre. Auch die Ergebnisrückgabe findet innerhalb der Menustruktur statt.

Fazit Aufgrund der mangelhaften Integrierbarkeit ist WebGlimpse nicht geeignet, ohne große Umstände als Suchsystem für OPUS/TUBdok genutzt zu werden.

[1] `http://webglimpse.net/`

3.2.8 Alkaline

Über die Software Alkaline[1] von der Schweizer Firma Vestris ist ein vollständiger Search- und Indexserver, der zwar Closed-Source, aber dennoch für Nichtkommerzielle Unternehmen kostenfrei erhältlich und nutzbar ist.

Entwicklungsaktivität Die Entwicklung ruht seit 2004 (Stand: Juni 2007).

Installation Zur Installation von Alkaline müssen lediglich die Binaries heruntergeladen und entpackt werden. Danach kann der Server in Betrieb genommen werden. Im Lieferumfang inbegriffen ist ein Demoverzeichnis mit Testdokumenten, über das die Lauffähigkeit der Software getestet werden kann. Beim Versuch, den Server zu starten, geriet dieser aber in eine Endlosschleife: der Hauptprozess stürzte ab und startete sich sofort neu.

Fazit Da die Ursache der ständigen Abstürze nicht zu ermitteln war, wurden weitere Untersuchungen mit Alkaline nicht unternommen.

3.2.9 Lucene

Über die Software Lucene[2] ist keine Suchmaschine, sondern eine Suchmaschinentechnologie. Mit Lucene können Indizes aufgebaut werden und Suchanfragen auf diese abgesetzt werden. Primär ist Lucene für Java gedacht, inzwischen gibt es aber zahlreiche Portierungen in andere Sprachen wie C++, C#, Perl oder Python. Nutch[3] ist eine Implementierung der Lucene-Technologie in Java.

PHP-Bindung Es gibt eine Implementierung für PHP 5.x, die über das Zend-Framework[4] verfügbar ist.

Installation Um sich den realistischsten Eindruck zu Lucene in PHP bilden zu können, wurde das Zend-Framework auf dem TUBdok Testsystem installiert und anschließend anhand der Beschreibungen in [Zen06] ein Index sowie eine provisorische Suchmaske in PHP implementiert.

[1] `http://alkaline.vestris.com/`
[2] `http://lucene.apache.org/`
[3] `http://lucene.apache.org/nutch/`
[4] `http://framework.zend.com/manual/en/zend.search.lucene.html`

Indexerzeugung Der selbst geschriebene Indexer und die Struktur des Index wurden bereits provisorisch an das TUBdok-System angepasst. So konnten den Indexfeldern direkt die Inhalte aus der TUBdok-Datenbank zugewiesen werden, anstatt die Metadaten aus dem PDF auszulesen, wie es bei den vorformulierten Suchsystemen die Regel war.

Testrecherche Die Testrecherche auf dem TUBdok-System ergibt 20 Treffer bei der Suche nach "Bernhard Sell". Lucene verwendet standardmäßig eine ODER-Verknüpfung bei mehreren Suchbegriffen. Da die statischen Indexdateien beim Indizieren übergangen wurden, um eine Doppelerfassung der Dokumente zu vermeiden, kann diese für ID 38 nicht gefunden werden. Das PDF zur ID 38 wird aber gefunden und mit der höchsten Relevanz gerankt. Alle anderen erwarteten Treffer sind ebenfalls in der Ergebnismenge enthalten. Beim Test mit einer expliziten UND-Verknüpfung werden die erwarteten fünf Treffer gefunden.

Fazit Lucene bietet beste Integrationsmöglichkeiten und trotz der Quick-and-Dirty-Implementierung zum Testen ein qualitativ sehr gutes Suchergebnis. Eine Verfeinerung des Suchsystem ist beliebig möglich. Daher ist eine Benutzung von Lucene für diese Arbeit sehr sinnvoll.

3.2.10 Xapian

Über die Software Xapian[1] ist ähnlich wie das im Abschnitt 3.2.9 vorgestellte Lucene eine Softwarebibliothek, die Methoden zum Erstellen einer eigenen Suchmaschine zur Verfügung stellt. Die Software steht unter GPL-Lizenz und ist frei verfügbar.

Entwicklungsaktivität Eine Weiterentwicklung findet noch aktiv statt, die letzte stabile Version ist derzeit Verion 1.0.1 vom 11.06.2007 (Stand: Juni 2007). Zum Ausprobieren der Leistungsfähigkeit von Xapian steht eine Beispielimplementierung namens Omega zur Verfügung.

PHP-Bindung Eine Bindung an PHP, um die Xapian API auch aus PHP anzusprechen, ist verfügbar.

Installation Die Kompilierung von Xapian scheierte auf Anhieb aufgrund von Fehlern im Bereich Flint[2]. Es gibt jedoch die Möglichkeit, Flint aus dem Kompi-

[1] `http://www.xapian.org`

[2] Flint ist der Name der Datenbank, die Xapian zum Halten der Indizes benutzt.

liervorgang auszuschließen und die Vorgängerdatenbank namens Quartz stattdessen zu nutzen. Eine Kompilierung mit Quartz funktionierte. Auch Omega konnte problemlos auf diese Xapian-Installation aufgesetzt werden. Die Kompilierung des PHP-Bindings klappte allerdings nicht, offenbar aufgrund der Auslassung von Flint.

Indexerzeugung Die Indexerzeugung wurde mit Hilfe der Omega-Installation vorgenommen. Der Xapian-Index für Omega kann mit Hilfe des Kommandozeilen-Tools *omindex* erstellt werden. Der Indexer läuft gegen ein einstellbares Verzeichnis auf der lokalen Festplatte. Als Parameter kann die Datenbank eingestellt werden, in der die Indexeinträge gespeichert werden, sowie ein URL, der anstelle des absoluten Dateipfades bei den indizierten Dokumenten erfasst werden soll. Die Ergebnisausgabe kann über Templates gesteuert werden.

Testrecherche Für die Recherche stellt Omega ein CGI-Programm zur Verfügung. Die Suchanfrage nach "Bernhard Sell" ergibt genau die erwarteten sechs Treffer. Es gibt auch die Möglichkeit einer exakten Phrasensuche, die beim Suchstring "Bernhard Sell" die erwarteten zwei Treffer ermittelt.

Fazit Allein das Scheitern der Kompilierung der PHP-Bindings spricht gegen den Einsatz von Xapian.

3.2.11 Weitere Systeme

Im Laufe der Evaluation der einzelnen Suchmaschinen wurden auch die folgenden Engines zunächst in Betracht gezogen, aber bereits die oberflächliche Betrachtung ergab, dass sie im Kontext der Abeit nicht zu nutzen sind:

- Fast[1]
  Fast ist eine Suchmaschinentechnologie zum Aufbau eigener Suchmaschinen, im Gegensatz zu Lucene aber Closed Source.

- Suma-dt
  Suma-dt ist die dem Forschungsportal[2] zugrundeliegende Software. Sie wurde am RRZN der Universität Hannover entwickelt. Die Software ist Closed Source (vgl. [Eic06], S. 81f).

[1] `http://www.fastsearch.com/`
[2] `http://forschungsportal.net`

3.3 Entscheidung

Wie aus den einzelnen Testberichten aus Abschnitt 3.2 ersichtlich, kamen nur mnoGoSearch, Namazu und Lucene für eine Vertiefung im Rahmen dieser Arbeit in Frage.

Der genaue Aufbau des Index ist bei mnoGoSearch und Namazu bereits vorgegeben und die Möglichkeiten zur Einflussnahme ohne einen Eingriff in die Quelltexte der Programme halten sich in Grenzen. Da Lucene wie beschrieben eine Suchmaschinentechnologie ist, wird keine statische Indexstruktur vorgegeben, sondern es muss eine eigene Struktur definiert werden. Der Vorteil dabei ist, dass die in der Datenbank vorgehaltenen Metadaten wesentlich genauer in die einzelnen Indexfelder sortiert werden können als bei den vorgefertigten Suchmaschinen. Der Nachteil ist, dass wesentlich mehr Implementierungsarbeit selbst geleistet werden muss.

mnoGoSearch verwendet eine MySQL-Datenbank als Backend zur Datenhaltung. Eine relationale Datenbank wie MySQL ist ein Werkzeug zur Datenstrukturierung und hat Funktionen, die von reinen Suchmaschinen nicht benötigt werden. Lucene dagegen verwendet ein eigenes Indexsystem zur Datenhaltung, das auf die Belange eines reinen Suchsystems optimiert ist und den Overhead relationaler Datenbankfunktionalitäten nicht bieten muss. Daraus folgt, dass für die Realisation eines rein auf Suche optimierten Systems eine Technologie wie Lucene einer MySQL-basierten Technologie vorzuziehen ist.

Die Integration von Lucene in PHP ist direkt über das Zend-Framework mit PHP5 möglich. Die meisten OPUS-Anwender haben derzeit noch PHP4-Systeme und auch TUBdok selbst wird produktiv auf einem PHP4-System betrieben. Inzwischen hat PHP das Ende der PHP-4-Produktlinie angekündigt (vgl. [PHP07a]). Daher soll auch der OPUS-Software kurz- bis mittelfristig auf die neue Version migriert werden. Mit dem Einsatz eines Lucene-basierten Suchsystems steigt die Anforderung an den OPUS-Server also auf PHP 5.x.
Da die Gestaltungsmöglichkeiten im Fall von Lucene am größten sind, soll analog zur ursprünglichen Planung dieser Arbeit das Suchsystem Lucene verwendet werden.

Kapitel 4

Suchsystem für OPUS

Zur Planung des Suchsystems wurde ein Modell in UML erstellt, in dem die zu implementierenden Klassen auf Seiten von OPUS dargestellt und beschrieben wurden. Soweit direkt Klassen aus dem Zend-Lucene-Paket genutzt wurden, wurden auch diese im UML-Diagramm aufgenommen. Bei der Planung wurden insbesondere die zur Anwendung kommenden Entwurfsmuster berücksichtigt.

4.1 Grundsätzliche Überlegungen

4.1.1 Codierung

OPUS arbeitet derzeit standardmäßig intern mit ISO-8859-1-Codierung, allerdings verwenden die vom KOBV (Kooperativer Bibliotheksverbund) betriebenen OPUS-Instanzen bereits UTF-8. Lucene benutzt standardmäßig UTF-8. Um das Suchsystem möglichst universell einsetzbar zu halten, ist es sinnvoll, die Codierung flexibel zu halten und über eine Config-Datei einstellbar zu machen. Solange OPUS auf ISO-8859-1-Codierung läuft, muss darauf geachtet werden, dass das Suchergebnis bei der Darstellung innerhalb von OPUS auch in diesem Zeichensatz codiert wird. Da der Index in UTF-8 gehalten wird, muss die Suchanfrage auch in dieser Codierung auf das System geschickt werden.

4.1.2 Kopiergeschützte PDFs

Kopiergeschützte PDF-Dateien können ein Problem für den Indexer darstellen. Der Indexer muss den Text aus dem Dokument extrahieren, um ihn zu indizieren. Dies ist jedoch nicht möglich, wenn der Text oder Teile des Textes z.B. durch Maßnahmen des DRM (Digital Rights Management) nicht kopiert werden können. Ein auf TUBdok publiziertes und auf diese Weise geschütztes Werk ist die Datei rep89.pdf

aus Dokument ID 158[1]. Der Text dieses Dokuments kann im Readerprogramm zwar markiert, aber nicht in die Zwischenablage übernommen werden. Viele Dokumente auf TUBdok unterliegen einem derartigen Kopierschutz. Um zu testen, ob die derart geschützten Dokumente mit dem provisorischen Indexer indiziert wurden, wurde aus dem o.g. Text das Wort "Drazin" gesucht. Tatsächlich ergibt die Suche nur einen Treffer, die ID 48. Das Dokument mit der ID 158 wurde demzufolge nicht korrekt indiziert.

Eine Lösung für dieses Problem wäre, die Autoren aufzufordern, ihre Dokumente nicht kopiergeschützt einzureichen. Derartige Anforderungen können in den Leitlinien zur Publikation auf dem Dokumentenserver festgelegt werden und würden dem Gedanken, die Dokumente OA anzubieten, entsprechen. DRM-Maßnahmen und Kopierrestriktionen bedeuten eine Einschränkung der Nutzbarkeit und demzufolge können solche Dokumente per definitionem nicht mehr OA sein. Ob der Dokumentenserverbetreiber derartige Nicht-OA-Dokumente zulässt, liegt im eigenen Ermessen, denn es ist nicht Voraussetzung, dass ein OA-Server zu 100% OA-Volltexte enthalten muss. Im Fall von TUBdok ist es derzeit möglich, auch Nicht-OA-Publikationen einzustellen, praktisch genutzt wurde dies aber noch nicht. Alle Dokumente sind zumindest generell offen zugänglich, auch wenn sie teilweise auf die beschriebene Weise kopiergeschützt sind.

Eine andere Lösung wäre es, den Indexer in solchen Fällen das Originalformat indizieren zu lassen. Dies setzt allerdings voraus, dass der Autor das Originalformat zur Verfügung stellt, was nur in seltenen Fällen passiert.

4.1.3 Indizierungstheorie in Lucene

Grundsätzlich wird der Prozess des Indexaufbaus als Indizierung bezeichnet. Zu indizierende Texte werden in der Regel zunächst analysiert. Bei der Analyse werden die Texte in Tokens zerlegt, über die anschließend weiter entschieden wird, ob bzw. wie sie indiziert werden sollen. Lucene hantiert dabei ausschließlich mit Plain-Text als Eingabetext; liegt die zu indizierende Datei nicht in reinem Text vor, so muss sie zunächst konvertiert werden. Bei der Analyse können Stoppwort-Listen definiert werden, die Worte beinhalten, die nicht im Index erfasst werden müssen. Auch muss entschieden werden, in welcher Form die Worte indiziert werden sollen. Soll z.B. die Suche nicht case-sensitiv arbeiten, können alle Worte in Kleinbuchstaben umgewandelt werden (vgl. [GH05], S. 19f).

In den Index übernommen werden letztlich Lucene-Dokumente. Diese sind in ver-

[1] `http://doku.b.tu-harburg.de/volltexte/2006/158/pdf/rep89.pdf`

schiedene Felder unterteilt. In den Feldern stehen die zu indizierenden Daten und Metadaten des Dokuments. Welche Felder in ein Lucene-Document übernommen werden, bleibt dem Implementierenden selbst überlassen. Jedem Feld muss ein Typ zugewiesen werden. Lucene unterstützt dabei verschiedene Feldtypen (vgl. [GH05], S. 21f):

- Keyword
  Dieser Feldtyp wird beim Indizieren keiner Analyse unterzogen und wird als Feldinhalt komplett im Urzustand gespeichert. Somit sind Keyword-Felder durchsuchbar, da sie indiziert sind.

- UnIndexed
  Dieser Feldtyp wird weder analysiert noch indiziert, aber im Originalzustand abgespeichert. Dies kann zur Ausgabe bestimmter Inhalte verwendet werden, die aber selbst nicht durchsuchbar sind.

- UnStored
  Dieser Feldtyp wird zwar analysiert und indiziert, aber nicht im Urzustand abgespeichert. So kann dieser Inhalt nicht zusammenhängend im Suchergebnis ausgegeben, aber trotzdem bei einer Suche gefunden werden.

- Text
  Dieser Feldtyp wird analysiert und indiziert und, wenn es ein Text ist, auch gespeichert. In diesem Feldtyp gespeicherte Inhalte können im Ursprungsformat im Suchergebnis ausgegeben werden.

4.1.4 Indexstruktur für die OPUS-Suche

Jean-Noël Jeanneney kritisiert, dass Googles Buchsuche dokumentenorientiert anstatt werksorientiert vorgeht.

> Bücher [müssen] »kontinuierlich und kumulativ« gelesen und verarbeitet werden ... Google wird diesem Bedürfnis jedoch nicht gerecht: Jedenfalls stehen dort momentan nur einzelne Seiten und nicht das Werk im Mittelpunkt, das in seiner Gesamtheit betrachtet werden muß.
> Auf Seiten hinzuweisen ist etwas völlig anderes, als auf Werke hinzuweisen. ([Jea06], S. 35)

Im Fall der Suchmaschine für OPUS ist es möglich, eine werksbasierte Suche zu realisieren, da sich jedes Dokument eindeutig einem Werk zuordnen lässt. Ein Werk in OPUS kann aus mehreren einzelnen Dateien bestehen. Mit dem werksbasierten

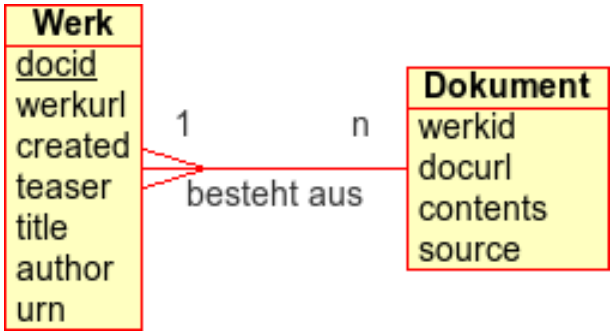

Abbildung 4.1: Entity-Relationship-Diagramm der Indexstruktur

Index kann im Suchergebnis bereits der Gesamttitel des Werks angezeigt werden und nicht nur ein einzelnes, womöglich aus dem Zusammenhang gerissenes, Dokument.

Probleme der werksbasierten Indizierung Jedes Dokument wird einzeln indiziert, damit im Suchergebnis erkennbar ist, welches Dokument des Gesamtwerkes auf die Suchanfrage passte. Dennoch muss erkennbar bleiben, zu welchem Werk das einzelne Dokument gehört. In einer relationalen Datenbank könnte diese Beziehung wie in Abb. 4.1 dargestellt aussehen. Lucene ist jedoch keine relationale Datenbank. Es ist zwar möglich, mehrere Indizes zu konstruieren, jedoch wäre dies mit einem großen Zusatzaufwand verbunden, weil dann nicht nur zwei Indizes gepflegt werden, sondern beide auch bei der Suche berücksichtigt werden müssten. Aus diesem Grund wurde entschieden, die Realisation in Lucene nur mit einem einzelnen Sammelindex zu realisieren, der sowohl die werksbezogenen, als auch die dokumentbezogenen Daten enthält.
Die Konsequenz aus dieser Vorgehensweise ist wiederum, dass jedes Werk, dem mehrere Dateien zugeordnet sind, auch mehrfach im Index verzeichnet ist. Dadurch wird die zweite Normalform relationaler Datenmodelle verletzt. Diese lautet: „Immer dann, wenn sich Inhalte in Spalten wiederholen, müssen die Tabellen in mehrere Teiltabellen zerlegt werden.“ ([Kof01], S.143). Der sich wiederholende Inhalt sind die zum Werk gehörenden Daten. Die Verletzung dieser Normalform wurde bewusst in Kauf genommen, da die parallele Pflege zweier Indizes in einem System, das an sich keine Relationenbildung unterstützt, als zu aufwändig angesehen wurde.

Feldaufteilung Wie im Abschnitt 4.1.3 dargestellt wurde, ist ein Lucene-Index in verschiedene Felder aufgeteilt, die unterschiedliche Metadaten zum Werk aufnehmen. Für die zu designende Suchmaschine wurde zunächst überlegt, welche Metadaten im Index erfasst werden und in welchen Feldtypen sie gespeichert werden sollen.

Jede Ressorce hat in OPUS mehrere URLs. Einer davon führt zur statischen, von OPUS erzeugten Indexseite des Werkes, einer führt zur dynamischen Indexseite, die ebenfalls dem Gesamtwerk zugeordnet ist. Außerdem hat jede zum Werk gehörende Datei einen eigenen URL, über den sie direkt aufrufbar ist. Im Index werden daher für jeden Eintrag zwei URL-Felder reserviert. Im Feld *werkurl* steht die dynamische Werksadresse, das Feld *docurl* enthält die Adresse des Einzeldokuments, das in diesem Eintrag indexiert ist.
Nach einem URL muss nicht gesucht werden können, da dieser auch ohne Suche direkt in die Adresszeile des Browsers eingegeben werden kann, wenn er bekannt ist. Daher muss der Inhalt des Feldes nicht indiziert werden, aber im Index zur Anzeige bereitstehen. Der Lucene-Feldtyp muss hier also *UnIndexed* sein.

Das Feld *created* sollte durchsuchbar sein, damit eine Sucheingrenzung auf Jahrgänge direkt vorgenommen werden kann. Es ist auch in der Ergebnisausgabe interessant, da es das Erstellungsdatum des Dokuments repräsentiert. Eine Analyse des Feldinhalts bei der Indizierung ist nicht notwendig, da in dem Feld nur ein Datum steht. Der Lucene-Feldtyp lautet daher *Keyword*.

Jedem Werk in OPUS wird ein Abstract zugewiesen, in dem der Inhalt des Werkes kurz und knapp zusammengefasst ist. Dieser soll in analysierter Form auch durchsuchbar sein, da er für die Suche interessante Stichworte enthalten kann. Der Abstract als gesamter Text sollte auch beim Suchergebnis angezeigt werden können, daher muss der Lucene-Feldtyp auf *Text* eingestellt werden. Der Feldname lautet *teaser*.

Das Feld *title* enthält den Titel der Arbeit, der sowohl im Suchergebnis als Werktitel angezeigt werden als auch suchbar sein soll. Daher wird auch der Titel als *Text* im Lucene-Index erfasst.

Im Feld *author* steht der Autor bzw. die Autoren des Werks. Hier gilt analog zum *title*, dass das Feld durchsuchbar sein muss und auch im Suchergebnis angezeigt werden soll. Daher wird auch hier der Lucene-Feldtyp *Text* verwendet.

Im Index soll auch, wenn möglich, der gesamte Volltext des Werkes erfasst werden. Dieser Volltext soll per Analyse und Tokenisierung durchsuchbar gemacht werden, daher ist eine Indizierung notwendig. Eine Anzeige des kompletten Volltextes im Suchergebnis ist allerdings nicht erforderlich und aufgrund des Umfangs dieser Texte auch nicht sinnvoll. Daher wird das Feld *content* als *UnStored* in den Index aufgenommen.

Feldname	Feldtyp
werkurl	UnIndexed
docurl	UnIndexed
created	Keyword
teaser	Text
title	Text
author	Text
contents	UnStored
docid	Keyword
urn	Keyword
source	UnIndexed

Tabelle 4.1: Indexstruktur des Lucene-Index für OPUS

Um gezielt überprüfen zu können, ob ein bestimmtes Dokument bereits indiziert ist, wird ein eindeutiges Merkmal benötigt. In der OPUS-Datenbank ist jedes Dokument durch die interne ID eindeutig gekennzeichnet. Dieser Wert wurde als Kennzeichner für das Werk in den Index übernommen. Da die ID ein reiner Zahlenwert ist, darf der Feldinhalt nicht analysiert werden, denn durch die Analyse werden Zahlenwerte ignoriert. Daher wurde der Feldtyp für das ID-Feld *docid* auf den nicht zu analysierenden Typ *Keyword* gesetzt.

Ein weiteres eindeutiges Merkmal für das Werk ist der URN. Da dieser aber nicht zwangsläufig für alle Dokumente vergeben wird, kann er nicht als zuverlässiger Ersatz für die ID im *docid*-Feld verwendet werden. Der Inhalt des *urn*-Feldes sollte suchbar sein können, um das Auffinden einer Ressource direkt über den URN zu ermöglichen. Als Feldtyp wurde daher auch hier *Keyword* gewählt.

Um bei der werksbasierten Suche noch zuordnen zu können, in welchem Einzeldokument der Suchtreffer erzielt wurde, muss auch die Quelle des indizierten Volltextes im Indexeintrag vermerkt werden. Das zu diesem Zweck angelegte Feld wird *source* genannt und enthält den Namen des indizierten Dokuments. Ist kein Volltext verfügbar, wird trotzdem ein Indexeintrag erzeugt, der in dem Fall nur die Metadaten des Eintrags aus der Datenbank enthalten kann. Bei solchen Einträgen wird im *source*-Feld daher der String *Metadaten* vermerkt. Das Feld soll nicht direkt durchsuchbar sein, aber im Suchergebnis angezeigt werden können. Als Feldtyp wird daher *UnIndexed* verwendet.

Die Indexstruktur ist in Tab. 4.1 als Übersicht zusammengefasst.

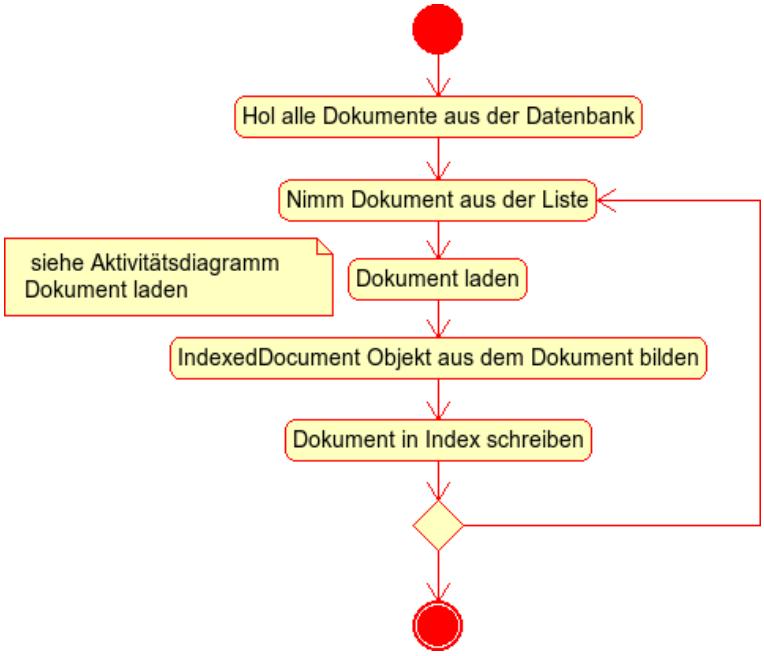

Abbildung 4.2: UML-Aktivitätsdiagramm zum Indizierungsvorgang

4.1.5 Indizierungsvorgang

In Abb. 4.2 ist erkennbar, wie die Indizierung abläuft. Der Vorgang muss in dieser Form nur einmalig beim ersten Aufbau des Index durchgeführt werden, da nur einmal sämtliche Einträge in der Datenbank durchgegangen werden müssen. Die folgenden Durchläufe des Vorgangs werden modifiziert und beziehen sich nur auf hinzugekommene Einträge: der Index wird nur noch erweitert.
Wie mit jedem einzelnen Dokument verfahren wird, ist in Abb. 4.3 dargestellt. Zunächst wird versucht, das Dokument im Präsentationsformat zu indizieren. Besteht das Werk aus mehreren Einzeldateien, wird jede einzelne Datei einem Indizierungsversuch unterzogen. Scheitert die Indizierung, wird überprüft, ob zu dem Werk ein Originalformat vorliegt. Ist eine Originaldatei vorhanden, wird versucht, diese zu indizieren. Scheitert auch das, wird das Werk ohne Volltextindizierung in den Index übernommen. Ein Eintrag muss für das Werk auf jeden Fall in den Index übertragen werden, damit das Werk trotzdem über die Suchmaschine gefunden werden kann. Dieser Eintrag setzt sich, wenn die Volltexte nicht indizierbar sind, ausschließlich aus Metadaten zusammen, die aus der Datenbank ausgelesen werden.

4.1.6 Ergänzen und Löschen von Einträgen

Nach dem einmaligen Initialisierungslauf des Indexers, bei dem die gesamte Datenbank indiziert wird, ist bei zukünftigen Indexerläufen eine Indizierung der bereits vorhandenen Einträge nicht mehr notwendig. Grundsätzlich gibt es zwei Möglichkei-

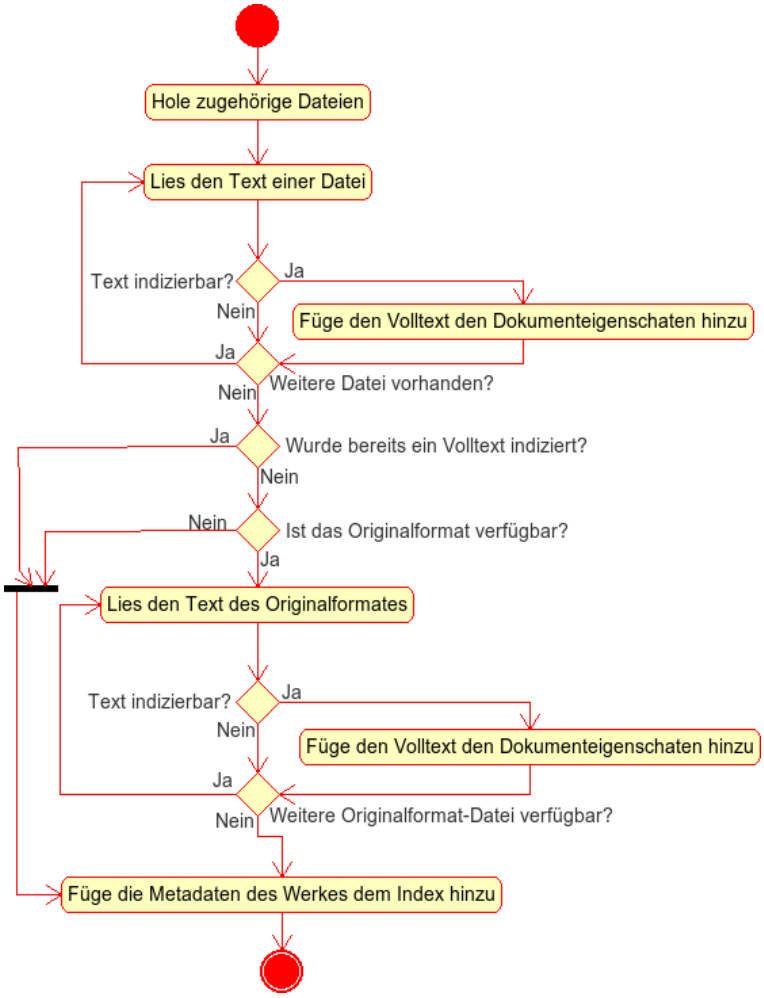

Abbildung 4.3: UML-Aktivitätsdiagramm zum Prozess des Einlesens eines Dokuments und der Registrierung des Volltextes

ten, den Index zu pflegen: durch periodischen Start des Indexer-Scriptes oder durch einen Start des Scriptes on demand.

Periodische Ausführung des Indexers Bei der periodischen Ausführung wird der Indexer in regelmäßigen Zeitabständen automatisch gestartet und überprüft bzw. optimiert seinen Datenbestand. Zur Realisation dieses Updates wurden zwei Vorgehensweisen getestet:

1. Durchgehen der gesamten Datenbank und Überprüfen, ob ein Eintrag bereits im Index erfasst ist
   Anhand einer eindeutigen ID, die sowieso jeder Eintrag in OPUS bekommt, kann der Indexer überprüfen, ob ein Eintrag aus der Datenbank schon im Index enthalten ist. Dazu dient das ID-Attribut, das (wie in Abschnitt 4.1.4 beschrieben) vom Typ Keyword ist und daher im Index auch suchbar ist. Nur wenn die ID nicht gefunden wird, indiziert der Indexer das Werk.

2. Selektive Suche in der Datenbank nur nach neuen Einträgen
   Diese Möglichkeit ist performanter als die erste genannte, jedoch bedingt sie, dass der Zeitstempel des letzten Indexerdurchlaufs und bei den Datensätzen in der Datenbank der Zeitstempel des Freischaltungszeitpunktes gesetzt werden müssen.

Mit beiden vorgeschlagenen Vorgehensweisen werden zum Update des Index nur Neueinträge gesucht. Auf diese Weise können im Laufe der Zeit tote Links entstehen, weil mit dem Entfernen eines Dokuments aus der Datenbank noch keine Entfernung aus dem Index verbunden ist. Das Indizierungssystem muss von der Löschung in Kenntnis gesetzt werden.

Beim Testen stellte sich heraus, dass in der Methode *Zend_Search_Lucene::find()* ein Fehler bei der Suche über Keyword-Felder vorliegt. Wird als Suchfeld gezielt *docid* angegeben und danach die gesuchte Nummer, so werden stets 0 Treffer zurückgegeben. Eine Suche mit Luke[1] ergab, dass die IDs ordnungsgemäß erfasst sind und auch innerhalb von Luke per Feldsuche gefunden werden können. Daher muss dieser Fehler in der PHP-Implementierung liegen. Dieser Fehler verhindert die ordnungsgemäße Suche nach Zahlen aller Art, wirkt sich also auch auf die Suche nach Jahrgängen aus, die für das *created*-Feld geplant war.

Nicht-periodische Ausführung des Indexers Bei einem nicht-periodischen Durchlauf können Ergänzung und Löschung von Einträgen aus dem Index auch

[1] Informationen zu Luke sind im Glorras zu finden.

synchron zum Geschehen auf dem OPUS-Server erledigt werden. Sobald ein Neueintrag stattfindet, wird der Indexer benachrichtigt und indiziert das neue Dokument sofort. Auch bei der Löschung erfolgt eine direkte Benachrichtigung, die den Indexer sofort dazu veranlasst, einen Eintrag zu löschen. Dies hat den Vorteil, dass Änderungen sofort sichtbar gemacht werden. Jedoch kann bei zahlreichen Neueinträgen die Gesamtleistung des Systems nachlassen. Generell ist es eher empfehlenswert, aufwändige Aktionen wie das Indizieren eines gesamten Datenbestandes zu besucherarmen Zeiten stattfinden zu lassen, in denen der Server seine Leistung dann dafür aufwenden kann und nicht noch parallel andere Benutzer bedienen muss.
Zum Löschen eines Eintrags muss zunächst die Lucene-interne ID des Eintrags ermittelt werden, was nur durch eine Suchanfrage auf den Index möglich ist. Aufgrund des Fehlers in der Feldsuche über bestimmte Felder ist ein Löschen von Dokumenten über die aus OPUS indizierte ID nicht möglich. Daher muss die Suchanfrage anders abgesetzt werden, aber trotzdem ein eindeutiges Ergebnis liefern, da ansonsten unbeabsichtigt Einträge aus dem Index entfernt werden könnten. Die Suchanfrage geht auf die Felder author mit dem kompletten Autorennamen und den vollständigen Publikationstitel. So muss das Ergebnis eindeutig sein.

4.1.7 Suche

Die selbst realisierte Suchmaschine sollte verschiedene Funktionen zum Suchen bieten. Welche Funktionen für die Suchsoftware im Einzelnen geplant waren und ob sie als optional oder obligatorisch daklariert wurden, ist in den folgenden Abschnitten beschrieben.

Werksbasierte Suche

Die Suche soll in erster Linie, wie bereits beschrieben, werksbasiert stattfinden. Das heißt, dass in der Ergebnisliste jedes gefundene Werk nur einmal angezeigt werden soll, auch wenn es aus mehreren Einzeldateien besteht, von denen jeweils mehrere den Suchbegriff enthalten. Auf diese Weise sollte ein übersichtlicheres Suchergebnis entstehen, in dem auch Bezüge der einzelnen gefunden Dateien erkennbar sind.

Suchen mit Trunkierung

Unter Trunkierung wird das Abkürzen von Suchbegriffen verstanden, um die Treffermenge dadurch zu erhöhen. Statt einem konkreten Wort wird der eigentliche

Suchbegriff durch ein Trunkierungszeichen (Wildcard) abgekürzt. Oft wird ein Fragezeichen oder ein Stern verwendet, wobei die beiden Zeichen auch unterschiedliche Bedeutung haben können (das Fragezeichen steht für einen einzelnen Buchstaben, der Stern für einen oder mehrere).
Trunkierung ist eine sehr sinnvolle Möglichkeit, kleine Treffermengen auszuweiten, wurde aber aufgrund der komplizierten Realisation zunächst nur als optional deklariert.

Phrasensuche

Phrasensuche ist die Suche nach mehreren Suchbegriffen in genau diesem Zusammenhang, um die Treffermenge dadurch zu verringern. In den meisten Suchmaschinen wird Phrasensuche in Form von Anführungszeichen angeboten: Begriffe, die direkt in diesem Kontext gefunden werden sollen, werden nacheinander in Anführungszeichen gesetzt.
Die Phrasensuche ist ein Hilfsmittel zur Eingrenzung großer Treffermengen, wurde aber aufgrund des kleinen Datenbestands auf TUBdok zunächst nur als optional deklariert.

Feldsuche

Wie im Abschnitt 4.1.4 beschrieben besteht ein Index aus verschiedenen Feldern. Über die Feldsuche kann eingestellt werden, in welchen dieser Felder der gesuchte Begriff vorkommen soll. So lässt sich eine Suchanfrage konkretisieren.
Die Eingrenzung auf verschiedene Felder ist eine Grundfunktionalität, um die Suche über bestimmte Metadaten realisieren zu können, daher soll von der Software obligatorisch unterstützt werden.

Klammerung bei der Suche

Eine Klammerung dient der logischen Aufgliederung einer Suchanfrage. Mit Klammerung können komplexe Suchanfragen wie z.B. *(Bibliothek or Universität) and Informationskompetenz* formuliert werden, um alle Dokumente zu finden, in denen Informationskompetenz im Kontext der Bibliothek oder der Universität vorkommt. Eine Klammerungsmöglichkeit wäre wünschenswert, ist aber aufgrund der komplizierten Realisierung auch als optional deklariert.

Suche mit Boole'schen Operatoren

Eine klassische Verfahrensweise zum Formulieren möglichst effizienter Suchanfragen besteht in der Nutzung boole'scher Operatoren (AND, OR, NOT). Sie dienen der

logischen Verknüpfung von Suchbegriffen. Mit AND verknüpfte Suchbegriffe müssen allesamt in den Dokumenten der Treffermenge vorkommen, bei der OR-Verknüpfung reicht es, wenn einer der verknüpften Begriffe im Dokument vorkommt. Über den NOT-Operator können gezielt Suchbegriffe ausgeschlossen werden, die in den gefundenen Dokumenten nicht vorkommen dürfen.
Die Unterstützung Boole'scher Operatoren kann als essentiell angesehen werden und wurde daher als obligatorisches Feature festgesetzt.

4.1.8 Ergebnisausgabe

Die Ergebnisausgabe sollte in verschiedenen Formaten stattfinden, von denen sich der Nutzer eines aussuchen kann. Standardmäßig soll bei der Suche per Browser ein HTML-Ergebnis zurückgegeben werden, das für einen menschlichen Nutzer am einfachsten lesbar ist und vom Browser interpretiert wird. Wird allerdings z.B. per Feedreader die Suchanfrage ausgelöst, so sollte die Ergebnisausgabe automatisch in RSS oder ASF stattfinden.
Welche Daten im Ergebnis aufgelistet werden, ist abhängig vom gewünschten Ausgabeformat. Schematisch soll die Ausgabe eines einzelnen Suchtreffers in XHTML so aussehen:
<Autor(en) (Nachname, Vorname; Nachname2, Vorname2; ...)>: <Titel in Originalsprache>
<Abstract (evtl. abgekürzt)>
<URL des Werkes> (Relevanz: <von Lucene berechnete Relevanz>)
gefunden in: <Dateiname>

4.1.9 Gesamtmodell

Das gesamte Indexsystem ist in Form eines UML-Klassendiagrammes in Abbildung A.1 dargestellt, das gesamte Suchsystem in Abbildung A.2. Die Abbildungen finden sich im Anhang A.

4.2 Modellierung und Entwurfsmuster

Ein Schwerpunkt dieser Arbeit liegt darin, die bei der Implementierung des Suchsystems angewendeten Entwurfsmuster zu benennen und zu beschreiben. Dabei wurden sowohl die direkt in der eigenen Umsetzung genutzten Entwurfsmuster, als auch diejenigen, auf die das Suchsystem Lucene baut, und die somit indirekt benutzt werden, berücksichtigt.

Als Grundlagenliteratur zur Identifikation und Beschreibung wurden im Wesentlichen [GHJV01] und [FFSB04] verwendet.

4.2.1 Singleton

Muster-Kurzbeschreibung

Zweck des Singleton-Musters ist folgender: „Sichere ab, daß eine Klasse genau ein Exemplar besitzt, und stelle einen globalen Zugriffspunkt darauf bereit.“ ([GHJV01], S. 157). Es darf also zu jedem Zeitpunkt nur maximal eine einzige Instanz einer bestimmten, Singleton implementierenden Klasse bestehen.

Einsatz des Musters

Der gesamte Indexer wurde als Singleton entworfen. Zu jedem Zeitpunkt t darf maximal eine Instanz des Indexers existieren. Würden zwei Instanzen parallel laufen, so könnte dadurch die Indexkonsisitenz gefährdet werden, weil von mehreren Instanzen aus schreibend auf den Index zugegriffen wird und nicht gewährleistet werden kann, dass nicht widersprüchliche Informationen eingetragen werden.
Die aus dem Zend-Framework stammende Klasse *Zend_Search_Lucene* ist als Singleton implementiert. Allerdings wird eine parallele Ausführung und damit mehrfache Instanziierung nur dann verhindert, wenn der Index neu erzeugt wird. Bei Aktualisierungen können auch mehrere Instanzen parallel gestartet werden, und dies geschieht dann nicht nach Singleton-Art. Im Fall der Neuerzeugung wirft *Zend_Search_Lucene* eine *Zend_Search_Lucene_Exception*, sobald eine weitere Instanz erzeugt wird, die auf den Index zugreift. Diese Exception kann bei der eigenen Implementierung abgefangen werden und als Erkennungssignal dienen.

Realisation

Eine Möglichkeit, ein Singleton-Muster zu realisieren, ist, beim Start einer Programminstanz eine Lock-Datei aufzubauen, durch die eindeutig gekennzeichnet wird, dass das Programm derzeit in Benutzung ist. Solange diese Lockdatei existiert, wird jeder weitere Versuch, ein Objekt der Klasse zu bilden, verhindert. Dies lässt sich zum Beispiel durch eine Überprüfung auf Vorhandensein der Datei im Konstruktor umsetzen. Auf diese Weise ist auch *Zend_Search_Lucene* realisiert.
Dies geht allerdings am ursprünglichen Ziel von Singleton noch etwas vorbei. Beim Singleton-Muster wird nicht unbedingt die parallele Nutzung der Instanz untersagt, sondern das Muster kann auch so genutzt werden, dass die Instanz als einzige Instanz für bestimmte Aktionen dient. So wird bei der Erzeugung eines Objektes überprüft,

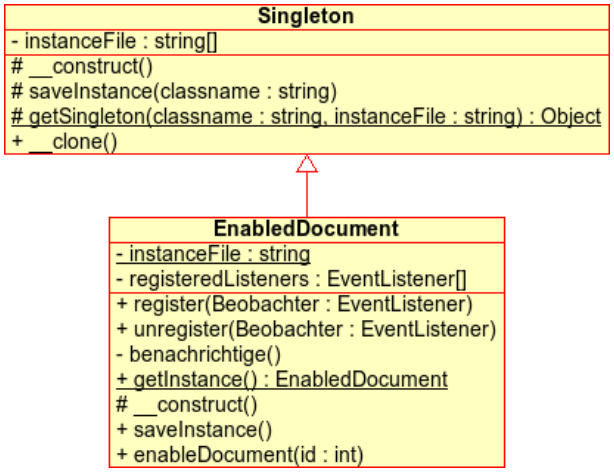

Abbildung 4.4: UML-Klassendiagramm der Singleton-Implementierung

ob bereits eine Instanz der Klasse existiert. Wenn dies der Fall ist, wird diese Instanz an die aufrufende Klasse zurückgegeben. Auf diese Weise existiert immer nur eine einzige Instanz eines Objektes, die aber verschiedene Aufgaben parallel erfüllen kann.

In Abbildung 4.4 ist in Form eines UML-Diagramms die grundlegende Singleton-Klasse dargestellt, die von allen konkreten Singleton-Klassen erweitert wird. Beispielhaft für die beerbende Klasse ist im Diagramm die Klasse EnabledDocument dargestellt. Weitere als Singleton implementierte Klassen sind im selbst entwickelten Indexsystem *DeletedDocument*, *Indexer* und *IndexerQueue*.

Bei der Implementierung eines Singleton wird der Konstruktor der Singleton-Klasse normalerweise als *private* deklariert und ist damit nur aus der implementierenden Klasse selbst zugreifbar. Da die in Abbildung 4.4 aufgeführte Singleton-Klasse als Schema für die untergeordneten Singleton-Klassen benutzt wird, ist eine Deklaration mit *private* nicht möglich. Um den abgleiteten Klassen, aber auch nur diesen, den Zugriff auf den Konstruktor __construct()[1] zu ermöglichen, wurde dieser als *protected*[2] deklariert. Auf diese Weise wird sichergestellt, dass nicht beliebige andere Klassen ein Objekt der Singleton-Klasse erzeugen können.
Andere Klassen können also über den Konstruktor keine Singleton-Objekte erzeugen. Um diesen anderen Klassen dennoch das Singleton-Objekt zur Verfügung stellen zu können, wird mit einer Klassenmethode[3] namens *getInstance()* (s. Abb. 4.4) gearbei-

[1] __construct ist eine universell einsetzbare Konstruktormethode in PHP5.

[2] Als *protected* deklarierte Methoden stehen der definierenden Klasse selbst sowie allen von ihr abgeleiteten Klassen direkt zur Verfügung.

[3] Klassenmethoden werden über das Schlüsselwort *static* deklariert und können aufgerufen werden, ohne dass ein Objekt der Klasse gebildet werden muss.

tet, die öffentlich zugreifbar ist. Über diese Methode wird der Zugriff auf die einzelne Instanz gesteuert: existiert noch keine Instanz, so wird eine neue angelegt und der aufrufenden Klasse zurückgegeben, andernfalls wird die existierende Instanz zurückgegeben. Dies hat gleichzeitig den Vorteil, dass das Objekt nur dann vorhanden ist, wenn es auch wirklich benötigt wird.

4.2.2 Beobachter (Observer)

Muster-Kurzbeschreibung

„Definiere eine 1-zu-n-Abhängigkeit zwischen Objekten, so daß die Änderung des Zustands eines Objekts dazu führt, daß alle abhängigen Objekte benachrichtigt und automatisch aktualisiert werden.“ [GHJV01], S. 287.

Einsatz des Musters

Die im Abschnitt 4.1.6 beschriebene Aktualisierungsroutine des Indexers wird nach der nicht-periodischen Vorgehensweise mit Hilfe des Beobachter-Musters umgesetzt. Das Indizierungssystem wird vom OPUS-System in Kenntnis gesetzt, wenn ein neuer Eintrag freigeschaltet bzw. wenn einer gelöscht wird. Das beobachtete Objekt ist also allgemein gesagt die OPUS-Datenbank, die Beobachter sind alle Objekte, die den Zustand der Datenbank abbilden. Das sind innerhalb von OPUS mehrere Objekte:

- die RSS-Feeds über die Neueinträge
- das Benachrichtigungssystem, das den Administrator über neue Einträge informiert bzw. den Autor über den Status seiner Anmeldung
- der Index

Im Rahmen dieser Arbeit ist nur der Index als Teil des Suchsystems relevant, die anderen Objekte bzw. Subsysteme können allerdings auch an den Beobachter andocken.

Realisation

Die Realisation des Beobachter-Musters in PHP ist etwas umständlich, da das klassische EventListener-Modell, das das Paradebeispiel für eine Beobachter-Implementierung ist, in PHP nicht direkt unterstützt wird. Es gibt keine EventListener- und Event-Klassen in PHP bzw. im Zend-Framework. Diese Klassen mussten für die Beobachter-Implementierung selbst geschrieben werden.
Das Auslösen der Aktualisierungsnachricht wird über den Aufruf einer speziellen

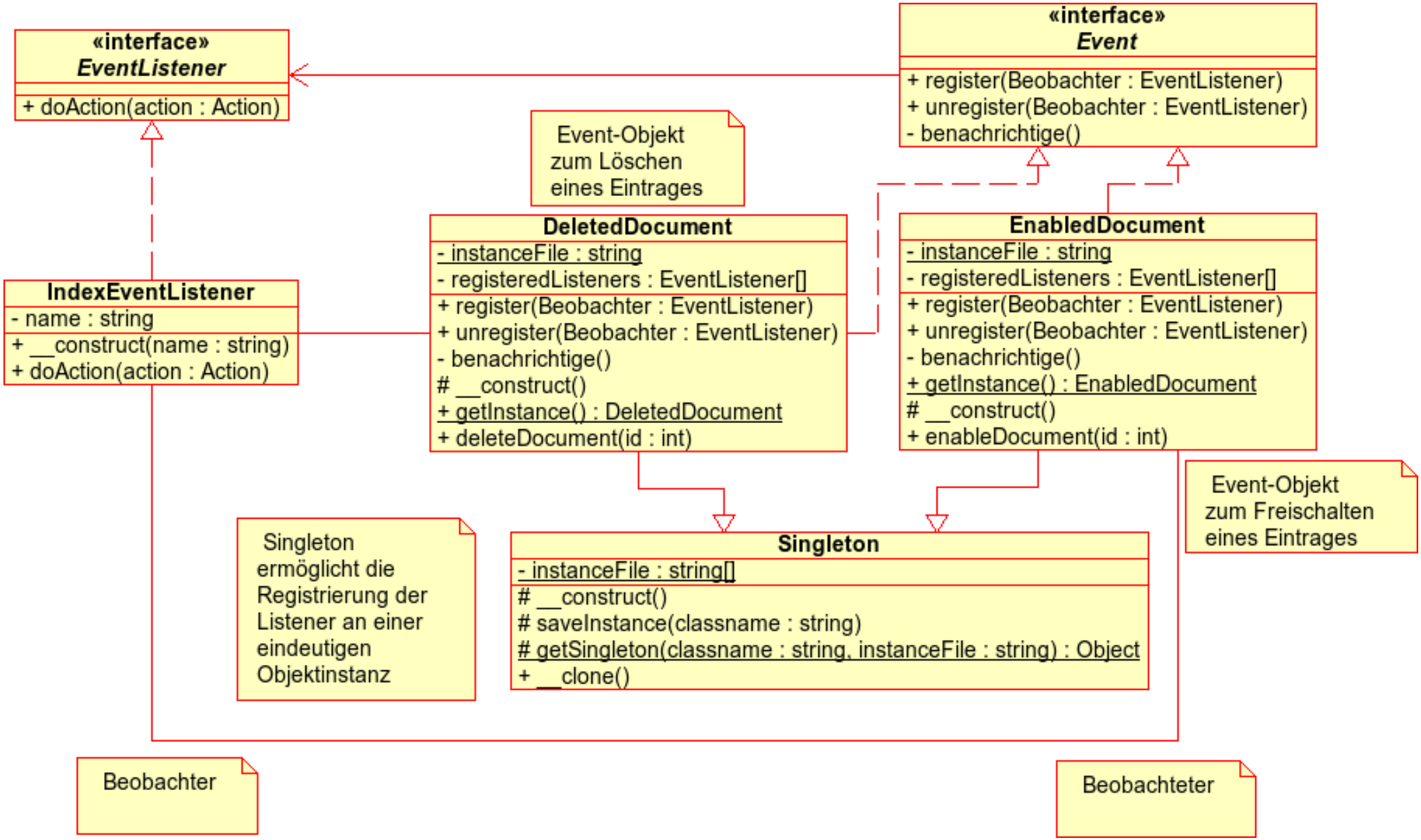

Abbildung 4.5: UML-Klassendiagramm der Beobachter-Implementierung

Methode der beobachteten Objekte (Eventobjekte) vorgenommen. Bei 1-zu-1-Beziehungen, also ein beobachtetes Subjekt und ein beobachtendes Objekt, könnte auch direkt eine Benachrichtigung des Beobachters ausgelöst werden. Allerdings bietet die Kapselung der Beobachter und der Beobachteten in eigene Klassen die Möglichkeit, das System leichter und ohne Änderung der auslösenden Aktion zu erweitern.

In Abbildung 4.5 ist die in der vorliegenden Arbeit verwendete Implementierung des Beobachter-Musters in Form eines UML-Klassendiagramms dargestellt.
Damit die EventListener-Objekte sich an einem eindeutigen Event-Objekt andocken können sind die Event-Objekte als Singleton realisiert (vgl. Abschnitt 4.2.1).
Als Beobachter agiert in der Abbildung der *IndexEventListener*, der die allgemeine *EventListener*-Klasse implementiert. *IndexEventListener* führt Aktionen aus, wenn Ereignisse eintreten. Die Ereignisse werden aus *Event* implementierenden Klassen ausgelöst. Im Diagramm sind zwei solche Klassen dargestellt: *EnabledDocument* und *DeletedDocument*. *EnabledDocument* ist für Mitteilungen gedacht, wenn ein neues Dokument freigeschaltet wurde. *DeletedDocument* wird ausgelöst, wenn ein Dokument in OPUS gelöscht wird.

4.2.3 Befehl

Muster-Kurzbeschreibung

„Kapsle einen Befehl als ein Objekt. Dies ermöglicht es, Klienten mit verschiedenen Anfragen zu parametrieren, Operationen in eine Queue zu stellen, ein Logbuch zu führen und Operationen rückgängig zu machen.“ [GHJV01], S. 273.

Einsatz des Musters

Im Beobachter-Modell wurden Beobachter-Objekte gebildet, die bestimmte andere Objekte sozusagen belauschen (siehe Abschnitt 4.2.2). Tritt ein bestimmtes Ereignis ein, so soll darauf eine Reaktion erfolgen. Um diese Reaktion flexibel zu gestalten, wurde bei der Implementierung auf das Befehlsmuster zurückgegriffen. Die auszuführende Aktion wird in ein eigenes Objekt gekapselt, das vom EventHandler verarbeitet werden kann.

Realisation

In Abb. 4.6 ist die Realisation im Indexer-Kontext in Form eines UML-Klassendiagramms dargestellt. Der Aufrufer des Befehls ist im vorliegenden Fall das EnabledDocument, das ein Action-Objekt als Befehl erstellt. Der konkrete

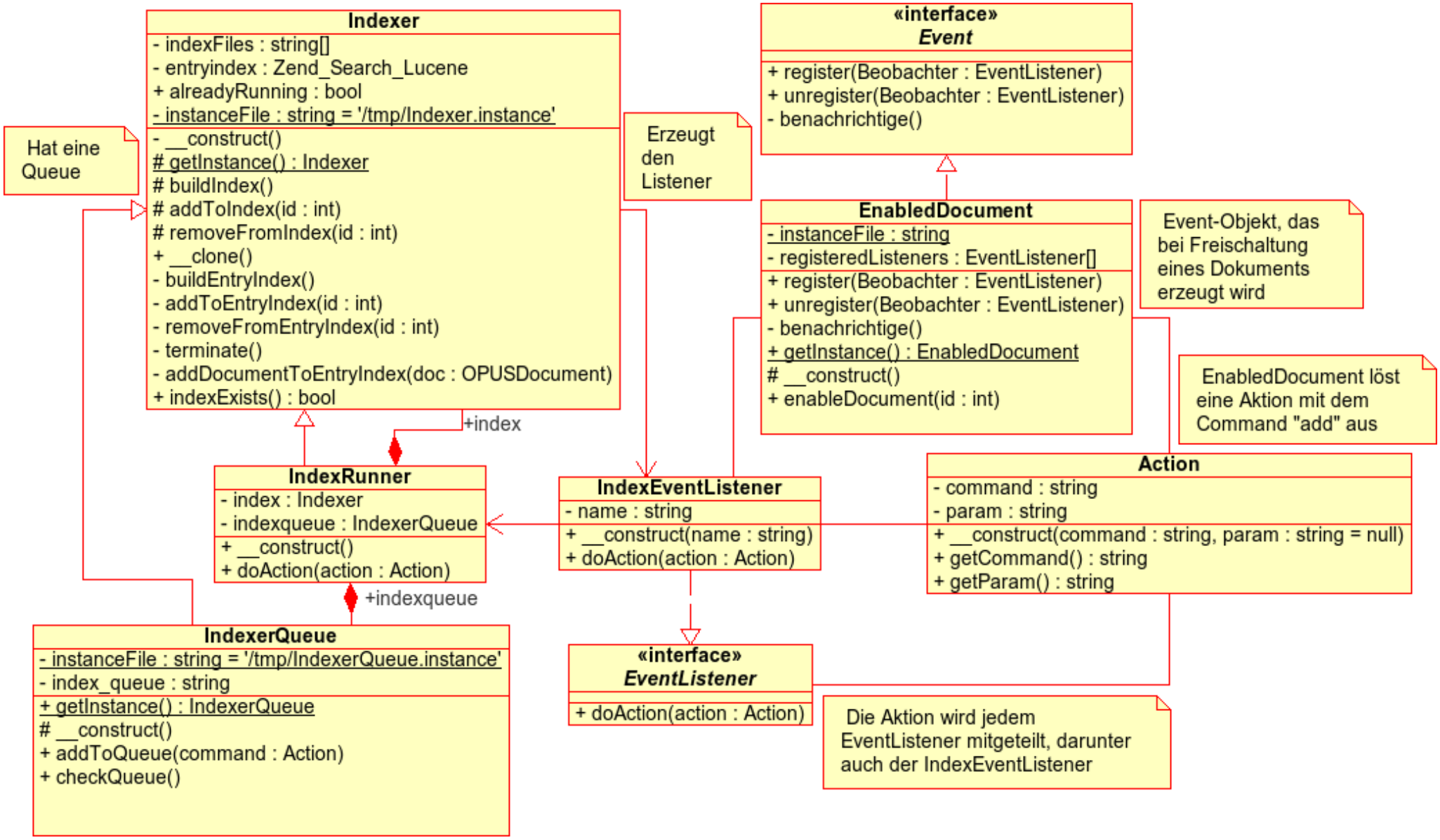

Abbildung 4.6: UML-Klassendiagramm der Implementierung des Befehl-Musters

Befehl ist in diesem Beispiel der Befehl *add*. Dieses Action-Objekt wird an alle Listener geschickt. In der Abbildung ist beispielhaft nur einer der Listener aufgeführt, der IndexEventListener. Die Listener implementieren die Methode *doAction()* und führen darüber den Befehl in einer für sie passenden Art und Weise aus.

4.2.4 Proxy

Muster-Kurzbeschreibung

„Kontrolliere den Zugriff auf ein Objekt mit Hilfe eines vorgelagerten Stellvertreterobjekts.“ [GHJV01], S. 254.

Einsatz des Musters

In Abb. 4.6 ist auch eine Klasse *IndexRunner* dargestellt. Diese Klasse dient dazu, Befehle, die an den Indexer geschickt wurden, zu verteilen. *IndexRunner* kann dabei als eine Art Proxy gesehen werden, da er dem eigentlich zu manipulierenden Indexer-Objekt, auf dem pro Zeiteinheit nur eine einzige Aktion möglich sein soll (vgl. Abschnitt 4.2.1), vorgelagert ist. Befehle werden nur sofort an den Indexer weitergegeben, wenn dieser nicht gerade mit einer anderen Aktion beschäftigt ist. Andernfalls wird der Befehl in eine Queue eingetragen und nach Beendigung der anderen Aktion automatisch ausgeführt.
Beim Proxy-Entwurfsmuster kann nach Gamma et al. zwischen verschiedenen Typen unterschieden werden. Unter anderem beschreiben Gamma et al. einen Schutzproxy, der den Zugriff auf das Originalobjekt schützt und eine Smart-Reference, die das eigentliche Objekt zunächst auf einen Lock überprüft (vgl. [GHJV01], S. 256).
Der hier verwendete Proxy kann als Mischung zwischen diesen beiden Anwendungsfällen beschrieben werden, da der *IndexRunner* den Zugriff auf das Originalobjekt absichert und gleichzeitig sicherstellt, dass nur ein Objekt zur Zeit Zugriff auf den Indexer hat. Alle anderen Aktionen, die der Indexer ausführen soll, landen in der Queue.

Realisation

In Abb. 4.7 ist die Realisation im Indexer-Kontext in Form eines UML-Klassendiagramms dargestellt. Das zu schützende Objekt ist dabei der Indexer, das modifizierende Subjekt ist die IndexerQueue, in der Kommandos lagern, die im Indexer ausgeführt werden sollen und der IndexRunner ist der dazwischenliegende Proxy. Der IndexRunner ist dabei eine Komposition von Indexer und IndexerQueue, enthält also selbst Instanzen dieser beiden Klassen. Dadurch kann IndexRunner

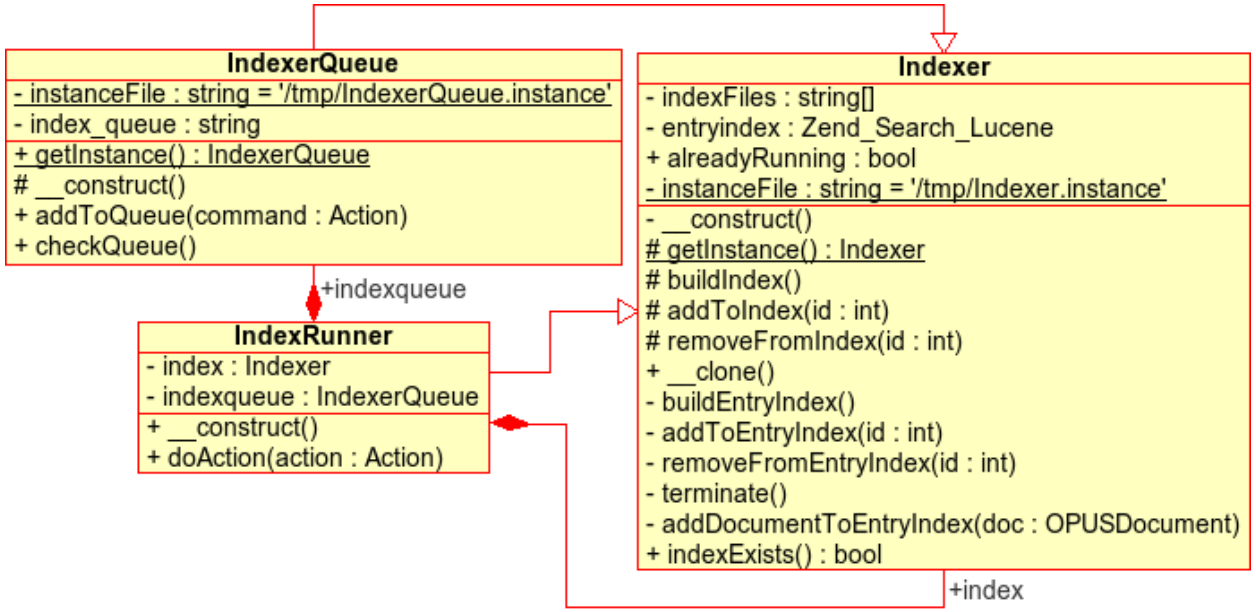

Abbildung 4.7: UML-Klassendiagramm der Implementierung des Proxy-Musters

ermitteln, welche der beiden Instanzen den Befehl weiterverarbeiten soll: ist der Indexer gerade beschäftigt, wird das Kommando in der Queue gespeichert, ist er frei, wird es direkt verarbeitet.

4.2.5 Strategie

Muster-Kurzbeschreibung

„Definiere eine Familie von Algorithmen, kapsele jeden einzelnen und mache sie austauschbar. Das Strategiemuster ermöglicht es, den Algorithmus unabhängig von ihn nutzenden Klienten zu variieren.“ [GHJV01], S. 373.
Das Strategie-Muster ist dem der Schablone (vgl. [GHJV01], S. 366) sehr ähnlich. Sie unterscheiden sich aber in einem Punkt: „Schablonenmethoden verwenden Vererbung, um die Teile eines Algorithmus zu variieren. Strategien ... verwenden Delegation, um den gesamten Algorithmus zu variieren.“ [GHJV01], S. 372. Aus diesem Grund werden beide im Folgenden beschriebenen Anwendungsbereiche im konkreten Fall dem Strategiemuster zugeordnet.

Einsatz des Musters

1. Indizierung verschiedener Volltext-Dateiformate
   Bei der Indizierung sollen verschiedene Dateiformate verarbeitet werden können. Die Dateien enthalten die Volltexte, die im Index erfasst werden sollen. Da Lucene nur Plain-Text verarbeitet, muss der Volltext zunächst in einen solchen reinen Text umgewandelt werden.

2. Bereitstellung verschiedener Rückgabeformate bei der Suche
   Bei einer Suchanfrage muss das Rückgabeformat eindeutig festgelegt werden. Da verschiedene Rückgabeformate angeboten werden sollen, können die einzelnen Formatimplementierungen als konkrete Strategien angesehen werden. Die allgemeine Strategie liegt in der Tatsache, dass das Suchergebnis zur Ausgabe formatiert werden soll. Über den auslösenden Kontext kann entschieden werden, welche konkrete Strategie angewendet werden muss.

Realisation

Beim Szenario 1 wird je nach Dateiformat der eigentliche Konvertierungsvorgang an eine Unterklasse delegiert, die entweder die Konvertierung selbst vornimmt oder ihrerseits die Rückgabe eines externen Konverters abwartet. Dieser Konverter bleibt austauschbar, falls zu einem späteren Zeitpunkt ein verbesserter Konverter zur Verfügung steht, der in das Programm integriert werden soll.

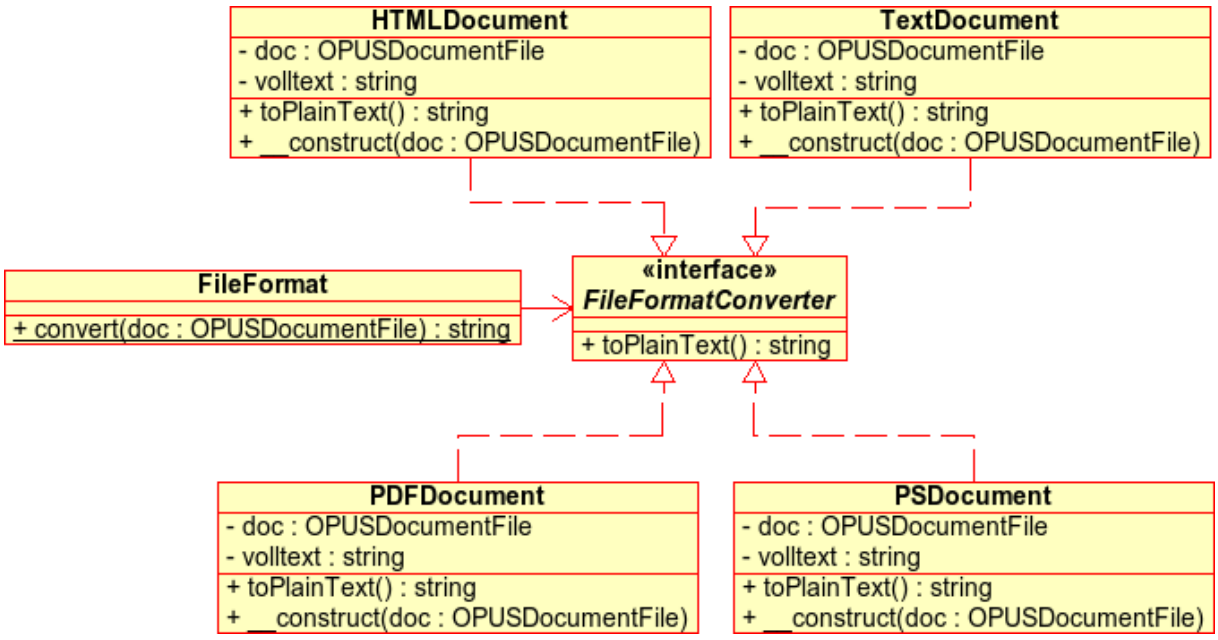

Abbildung 4.8a: UML-Klassendiagramm der Strategie- und Fabrikmethoden-Implementierung für die Indizierung

Für das Szenario 2 kann die passende Strategie per Parameterübergabe oder über eine automatische Erkennung des anfragenden Clients ausgewählt werden. Ist der anfragende Client ein Browser, so ist eine Rückgabe der Suchergebnisse als HTML sinnvoll, ist es ein RSS-Reader, so sollte gleich RSS zurückgegeben werden, entsprechend bei Atom-Readern ASF.
In den Abbildungen 4.8a und 4.8b sind die beiden beschriebenen Strategie-Implementierungen in Form von UML-Klassendiagrammen dargestellt. Mittelpunkt von Abbildung 4.8a ist das Interface *FileFormatConverter*, in der eine Methode definiert wird, die von allen implementierenden Klassen ausformuliert werden muss. Das Interface ist im Bild des Strategie-Musters die Strategie, alle die Strategie implementierenden Klassen sind Konkrete Strategien. Die zu konvertierende Datei ist der Kontext, denn sie hat ein Format, das in der Klasse FileFormat abgebildet wird. Je nach Format kann die passende Strategie ausgelöst werden.
In Abbildung 4.8b ist eine andere Art der Realisation abgebildet. Hier nimmt nicht ein Interface die zentrale Rolle ein, sondern eine abstrakte Klasse. In der abstrakten Klasse OutputFormat, die in diesem Fall die Strategie darstellt, werden abstrakte Methoden definiert, die in den Konkreten Strategie-Klassen ausformuliert werden. Außerdem werden in der Klasse einige allgemeingültige Methoden und Eigenschaften definiert, was beim Interface nicht möglich und nicht notwendig war. Auch der Kontext wird anders umgesetzt: das gewünschte OutputFormat wird aus einer Fabrik heraus erzeugt, die eine Komposition aus der konkreten

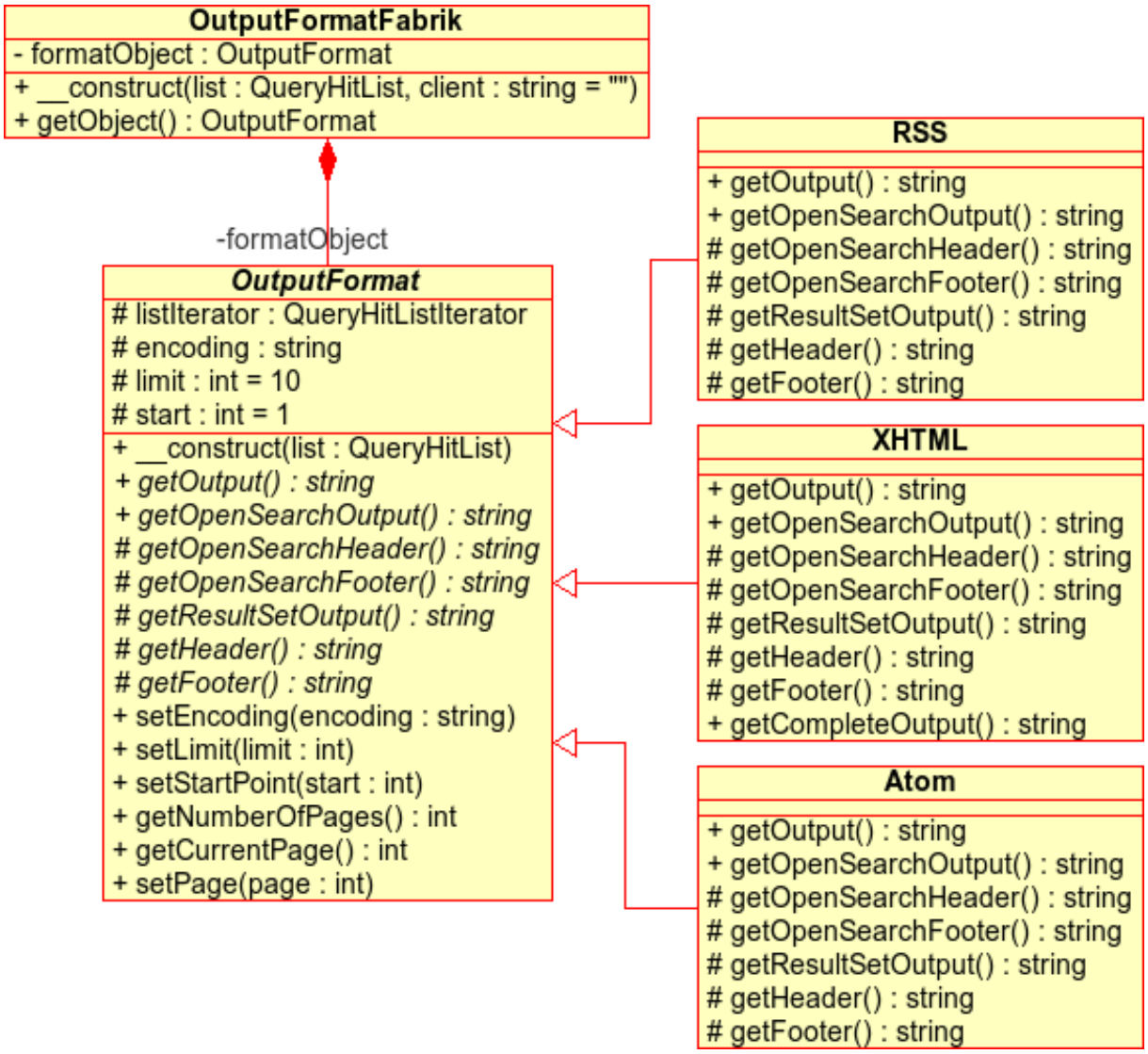

Abbildung 4.8b: UML-Klassendiagramm der Strategie- und Fabrikmethoden-Implementierung für die Suche

OutputFormat-Klasse bildet.

4.2.6 Fabrikmethode

Muster-Kurzbeschreibung

„Definiere eine Klassenschnittstelle mit Operationen zum Erzeugen eines Objekts, aber lasse Unterklassen entscheiden, von welcher Klasse das zu erzeugende Objekt ist. Fabrikmethoden ermöglichen es einer Klasse, die Erzeugung von Objekten auf Unterklassen zu deligieren.“ [GHJV01], S. 131.

Einsatz des Musters

Das Strategie-Muster wurde, wie im Abschnitt 4.2.5 beschrieben, in zwei Kontexten angewendet. Für das unter Punkt 1 des Abschnitts 4.2.5 dargestellte Szenario ist die Auswahl des Converters abhängig vom zu indizierenden Dateiformat. Über eine Methode wird entschieden, welches Objekt im Fall des zu indizierenden Dokuments gebildet werden muss.
Beim Szenario unter Punkt 2 des Abschnitts 4.2.5 kann durch Erkennung des anfragenden Clients oder über Parameterübergabe des Nutzers das passende Rückgabeformat zu einer Suchanfrage erstellt werden. Zur Erzeugung des Objektes, das den letztlich den gewünschten Output liefert, wird eine Fabrikklasse verwendet.

Realisation

Die Implementierung der Fabrikmethoden ist auch aus den Abbildungen 4.8a und 4.8b zu entnehmen.
Beim Szenario 1 liegt die Fabrikmethode in der Klasse *FileFormat* in der Methode *convert()*. Diese Methode entscheidet anhand des zu konvertierenden Dateiformates, welches Objekt auf der Schnittstelle *FileFormatConverter* aufgebaut werden muss, um den Volltext aus der Datei konvertieren zu können. Über die *toPlainText()*-Methode in der passenden Subklasse wird der Konvertierungsvorgang umgesetzt.
Für das Szenario 2 wird die Klasse, aus der das Objekt für das Rückgabeformat der Suche gebildet werden soll, in der Klasse *OutputFormatFabrik* ausgewählt. Als Produkt wird ein Objekt gebildet, das auf der Klasse *OutputFormat* basiert. Über die Zugriffsmethode *getOutput()* im *OutputFormatFabrik*-Objekt kann nach der Erzeugung des Objekts der passende Output abgeholt werden.

4.2.7 Adapter

Muster-Kurzbeschreibung

„Passe die Schnittstelle einer Klasse an eine andere von Ihren Klienten erwartete Schnittstelle an. Das Adaptermuster läßt Klassen zusammenarbeiten, die wegen inkompatibler Schnittstellen ansonsten dazu nicht in der Lage wären.“ ([GHJV01], S. 171). Der Adapter ist also ein Mittler zwischen verschiedenen Systemen, die zusammengeführt werden.

Einsatz des Musters

Das gesamte im Rahmen dieser Arbeit entwickelte Suchsystem ist eine Zusammensetzung schon vorhandener Bibliotheksbausteine und neu entwickelter, an das OPUS-System angepasster Klassen. Natürlich ist es möglich, die Klassen so zu gestalten, dass sie exakt auf das ausgewählte Suchsystem zugeschnitten sind und durch die vom Zend-Lucene-Paket bereitgestellten Schnittstellen optimal bedient werden. Jedoch ist es denkbar, dass das Suchsystem in Zukunft gegen ein anderes ausgetauscht werden soll. Es ist wahrscheinlich, dass ein anderes System über andere Schnittstellen angesprochen werden muss. Wäre das hier entwickelte Suchsystem so gestaltet, dass es nur an die Lucene-Klassen angepasst ist, dann müsste im Fall des Wechsels der Suchmaschine alles angepasst und ggf. neu implementiert werden. Um dies zu vermeiden, wurde das Adapter-Muster als Wrapper zwischen der eigenen Entwicklung auf OPUS-Seite und den von Zend bereitgestellten Klassen auf der Suchmaschinenseite zwischengeschaltet.
Das Adaptermuster hat viel Ähnlichkeit mit dem Brückenmuster. Beim Brückenmuster soll eine Schnittstelle von ihrer Implementierung getrennt werden (vgl. [GHJV01], S. 186ff). Dies ist im vorliegenden Fall nicht notwendig, da *Zend_Search_Lucene* bereits eine Implementierung der Suche anbietet, die im selbst erstellten System so übernommen werden kann. Das Brückenmuster wäre nur sinnvoll, wenn die Schnittstelle neu implementiert werden sollte.

Realisation

An allen Stellen, wo das OPUS-Suchsystem mit der bereitgestellten Suchmaschine kommuniziert, wurden Wrapper-Klassen eingeschaltet, die die Übersetzung von der OPUS-Schnittstelle auf die Suchmaschinenschnittstelle und umgekehrt übernehmen. Im Wesentlichen ist dies an zwei Stellen des Suchsystems der Fall: bei der eigentlichen Suchanfrage (dargestellt in Abb. 4.9a) sowie bei der Umwandlung der einzelner Suchtreffer vom Lucene-Format ins Format des eigenen Suchsystems (dargestellt in

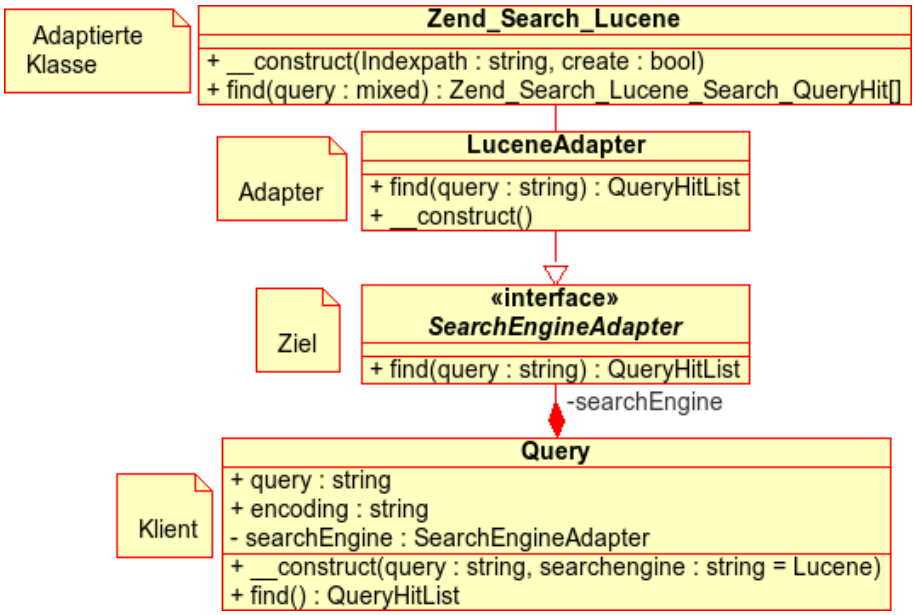

Abbildung 4.9a: UML-Klassendiagramm der Adapter-Implementierung für die Suche

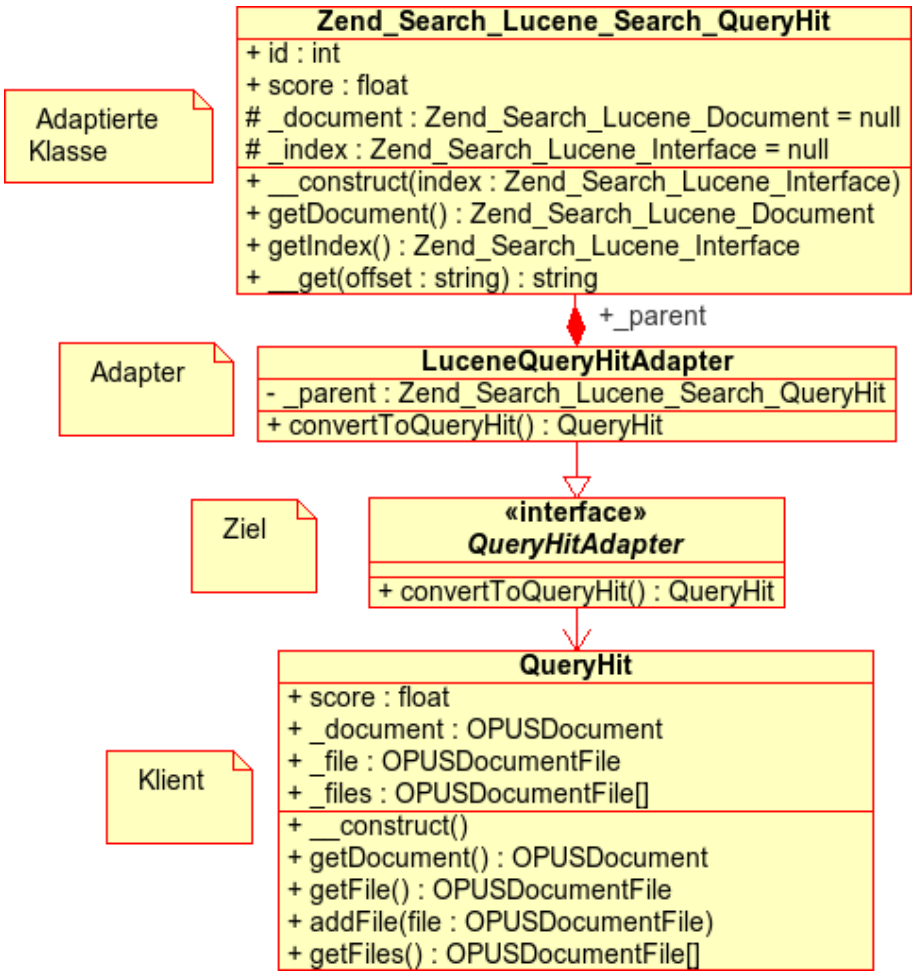

Abbildung 4.9b: UML-Klassendiagramm der Adapter-Implementierung für einzelne Suchtreffer

Abb. 4.9b).

Adapter bei Suchanfragen In Abb. 4.9a wird gezeigt, wie die in OPUS gestellte Anfrage in der *Query*-Klasse über den *LuceneAdapter* in eine Anfrage an *Zend_Search_Lucene* umgewandelt wird. Dabei komponiert *Query* im Attribut *searchEngine* die gewünschte Suchmaschine, so dass sie bei Bedarf an dieser Stelle einfach geändert werden kann. Der *LuceneAdapter* ist derzeit der einzige Adapter, der zur Verfügung steht. Alle Adapter müssen grundsätzlich die Schnittstelle *SearchEngineAdapter* implementieren, damit sie ordnungsgemäß von *Query* komponiert werden können.

Adapter bei Suchtreffern In Abb. 4.9b ist dargestellt, wie jeder einzelne Suchtreffer über einen Adapter in das OPUS-interne Suchtreffer-Format umgesetzt wird. Auf der OPUS-Seite des Suchsystems werden die Suchtreffer als *QueryHit*-Objekte repräsentiert, auf Lucene-Seite wird eine Repräsentation als *Zend_Search_Lucene_Search_QueryHit* vorgenommen. Der dazwischenliegende Adapter ist für Lucene der *LuceneQueryHitAdapter*, der die Lucene-interne Repräsentation des QueryHits in der Eigenschaft *_parent* komponiert. *LuceneQueryHitAdapter* implementiert die Schnittstelle *QueryHitAdapter*, die auch als Grundlage für andere Suchmaschinen-Adapter genutzt werden soll.

Unterschiede der Adapter-Implementierungen Anders als beim Adapter für die Suchanfrage wird die Komposition beim Suchtreffer-Adapter näher am adaptierten Suchsystem angesetzt. Der Grund dafür ist die verschiedene Art von Client, der den Adapter anspricht. Der *LuceneAdapter* wird direkt von der *Query* aktiviert und gibt eine für OPUS strukturierte *QueryHitList* zurück. Diese besteht aus einzelnen *QueryHit*-Objekten. Um die Liste erstellen zu können, muss der *LuceneAdapter* den *LuceneQueryHitAdapter* ansprechen und darüber die Konvertierung der einzelnen Suchtreffer veranlassen. Die beiden Adapter sind also untereinander verkettet, aber nur in einer Richtung. Da die zu nutzende Suchmaschine und damit implizit auch die zu nutzende Adapter-Klasse mit der *Query* ausgewählt wird, wird die Adapterklasse zum Konvertieren der *QueryHits* auch implizit durch die Wahl der Suchmaschine vorgegeben. Der *QueryHit* wird von dem *QueryHitAdapter* erzeugt, während es bei *Query* genau umgekehrt ist: sie erzeugt und verwaltet den *SearchEngineAdapter*.

4.2.8 Iterator

Muster-Kurzbeschreibung

„Ermögliche den sequentiellen Zugriff auf die Elemente eines zusammengesetzten Objekts, ohne seine zugrundeliegende Repräsentation offenzulegen.“ [GHJV01], S. 335.

Einsatz des Musters

Das Iterator-Muster wird im Zusammenhang mit der Suche angewendet. Das zusammengesetzte Objekt wird repräsentiert durch die Ergebnisliste, die alle auf die Anfrage passenden Treffer aus dem Index enthält. Mit dem Iterator wird eine einheitliche Schnittstelle zum Durchlaufen der Suchtrefferliste bereitgestellt, die unabhängig vom tatsächlichen internen Format der Liste ist und eine Weiterverarbeitung jedes einzelnen Suchtreffer ermöglicht.
Der Nutzen der Anwendung des Musters liegt in der formatunabhängigen Verarbeitung der Ergebnisliste. Sollte es notwendig werden, das interne Listenformat zu ändern, müsste durch die Verwendung dieses Entwurfsmuster tatsächlich nur die Liste an sich und der Iterator modifiziert werden, nicht aber sämtliche einzelnen Klassen, die auf die Liste zugreifen.

Realisation

Die für OPUS angefertigten Implementierung wird in Abbildung 4.10 als UML-Klassendiagramm dargestellt.
Für die Realisation wird über dem Konkreten Aggregat (*QueryHitList*) ein Iterator (*QueryHitListIterator*) gebildet, der die Ergebnisliste Element für Element durchlaufen kann. Ein Einzelelement ist dabei ein Suchtreffer-Objekt aus der Klasse *QueryHit*.

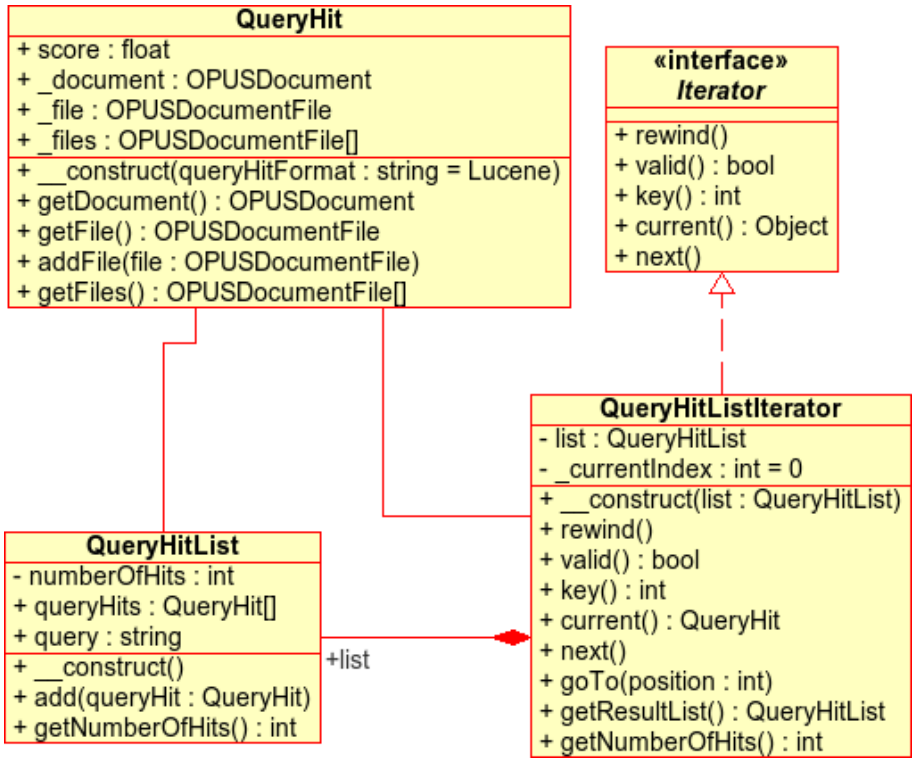

Abbildung 4.10: UML-Klassendiagramm der Implementierung des Iterator-Musters bei der Ergebnisliste

Kapitel 5

Information-Retrieval-Schnittstelle

5.1 Allgemeines zur Schnittstelle

Die Information-Retrieval-Schnittstelle soll dazu dienen, das realisierte Suchsystem auch für externe Systeme ansprechbar zu machen. Die Systeme sollen über eine standardisierte Schnittstelle Anfragen an das Suchsystem schicken können und als Antwort ein normiertes Format erhalten, das sie selbstständig weiterverarbeiten können. Durch dieses einheitliche Format wird das im Abschnitt 2.1.1 beschriebene Problem des Parsens der Suchtreffer gelöst. Das layout-unabhängige Rückgabeformat kann von Metasuchmaschinen wesentlich einfacher verarbeitet werden als der normalerweise ausgegebene HTML-Code.
Um diese Schnittstelle zu realisieren, wurde das XML-Format OpenSearch gewählt.

5.2 OpenSearch

5.2.1 Allgemeines

OpenSearch[1] ist ein Format, mit dem Suchtreffer und Ergebnislisten zwischen Suchdiensten ausgetauscht werden können. Das Format wurde von der Firma Amazon entwickelt, ist vollständig XML-basiert und offengelegt, so dass jeder Erweiterungen an dem Format vornehmen kann. Es wird bereits von einigen Suchmaschinen eingesetzt, z.B. vom Amazon-Ableger A9[2] oder vom experimentellen KeyWop[3]. Auch die in den Browsern Mozilla Firefox 2.0 und Internet Explorer 7 eingebaute Suchleiste wird über OpenSearch gesteuert (vgl. [Ope07]). Derzeit aktuellste Version des OpenSearch-Formats ist 1.1. Diese Version ist allerdings noch nicht endgültig ver-

[1] `http://www.opensearch.org/`
[2] `http://a9.com/`
[3] `http://keywop.com/`

abschiedet und liegt noch nur als Draft vor. In dieser Arbeit wird Version 1.1 in Draft 3[1] genutzt, da unterschiedliche Outputformate über OpenSearch ausgeliefert werden sollen. Diese Anforderung war mit Version 1.0 von OpenSearch noch nicht erfüllbar.

5.2.2 Struktur

OpenSearch besteht aus verschiedenen Komponenten (vgl. [Cli07d]).

- OpenSearch Description Document - beschreibt die Struktur der OpenSearch-konformen Suchmaschine
- OpenSearch Response - Antwort auf eine Suchanfrage

OpenSerach Description Document

Im Description Document wird die Suchmaschine beschrieben, z.B. der Name, welcher URL zum Abschicken der Suchanfrage benötigt wird und weitere relevante Metadaten.

Listing 5.1: Root-Element des OpenSearch Description Documents

```
1 <OpenSearchDescription xmlns="http://a9.com/-/spec/
    opensearch/1.1/">
2   <!-- ... -->
3 </OpenSearchDescription>
```

Das Root-Element eines OpenSearch Description Documents ist OpenSearchDescription. Die Definition der Description folgt daher dem in Listing 5.1 aufgezeigten Gerüst.

Listing 5.2: Url-Element im OpenSearch Description Document

```
1 <Url type="application/rss+xml" template="http://example.com
    /?q={searchTerms}&amp;pw={startPage?}" />
2 <Url type="application/xhtml+xml" template="http://example.
    com/search?q={searchTerms}&amp;start={startIndex?}" />
```

Ein sehr wichtiges Element in diesem Description-Document ist das *Url*-Element. In diesem Element wird definiert, welche Ergebnisformate die Suchmaschine ausliefert und unter welchem URL mit welchen Parametern der Anfragende die Ergebnisse abrufen kann. In Listing 5.2 ist ein Beispiel dargestellt. Zeile 1 zeigt die Adresse, unter

[1] Die Spezifikation zu OpenSearch 1.1.3 ist in [Cli07d] zu finden.

der ein RSS-formatiertes Ergebnis angefordert werden kann, Zeile 2 zeigt die Adresse für ein XHTML-Ergebnis. Die Suchanfrage selbst ist normalerweise nicht in eine XML-Struktur gekapselt. Die Parameter werden an das OpenSearch-Suchsystem über die im Description Document festgelegten URLs per HTTP GET an den Server gesendet. Im Beispiel wird der Suchstring einfach an Stelle des Platzhalters {searchTerms} gesetzt und dadurch als Parameter q an das Suchsystem geschickt.
Die komplette für TUBdok festgelegte Description-Datei ist in Anhang B abgedruckt.

OpenSearch Response Document

Das Response-Dokument enthält die zur Suchanfrage ermittelte Ergebnisliste. Der Nutzer kann ein beliebiges von der Suchmaschine unterstütztes Format für die Suchtreffer anfordern. Das gewünschte Format kann dabei als Parameter mit der Anfrage an das Suchsystem übergeben werden. Üblich sind folgende Formate:

- RSS
- ASF
- XHTML

Listing 5.3: OpenSearch Response in RSS 2.0

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <rss version="2.0"
3     xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/"
4     xmlns:atom="http://www.w3.org/2005/Atom">
5    <channel>
6      <title>Example.com Search: New York history</title>
7      <link>http://example.com/New+York+history</link>
8      <description>Search results for "New York history" at
           Example.com</description>
9      <!-- ... Open-Search-spezifische Elemente ... -->
10     <item>
11       <title>New York History</title>
12       <link>http://www.columbia.edu/cu/lweb/eguids/
             amerihist/nyc.html</link>
13       <description>
14         ... Harlem.NYC - A virtual tour and information on
               businesses ... with historic photos of Columbia
```

```
            's own New York neighborhood ... Internet
            Resources for the City's History. ..
15        </description>
16      </item>
17    </channel>
18  </rss>
```

Listing 5.4: OpenSearch Response in Atom

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <feed xmlns="http://www.w3.org/2005/Atom"
3        xmlns:opensearch="http://a9.com/-/spec/opensearch
           /1.1/">
4    <title>Example.com Search: New York history</title>
5    <link href="http://example.com/New+York+history"/>
6    <updated>2003-12-13T18:30:02Z</updated>
7    <author>
8      <name>Example.com, Inc.</name>
9    </author>
10   <id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af6</id>
11   <!-- ... Open-Search-spezifische und Atom-Meta-Elemente
       ... -->
12   <entry>
13     <title>New York History</title>
14     <link href="http://www.columbia.edu/cu/lweb/eguids/
         amerihist/nyc.html"/>
15     <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
16     <updated>2003-12-13T18:30:02Z</updated>
17     <content type="text">
18       ... Harlem.NYC - A virtual tour and information on
           businesses ... with historic photos of Columbia's
           own New York neighborhood ... Internet Resources
           for the City's History. ...
19     </content>
20   </entry>
21 </feed>
```

Listing 5.3 und 5.4 zeigen Beispiele für OpenSearch-Responses. Listing 5.3 demonstriert dabei die Rückgabe in RSS 2.0, Listing 5.4 liefert ASF. In beiden Beispielen ist der jeweilige Aufbau des Ergebnisdatensatzes dargestellt: in ASF ist jeder Suchtref-

fer in <entry> Elemente gekapselt, bei RSS heißt das Element <item>. Ansonsten sind die Rückgaben sehr ähnlich, bei ASF ist erkennbar, dass mehr Informationen im Ergebnis untergebracht werden können.
Je nach Rückgabeformat kann der Nutzer unterschiedliche Weiterverarbeitungswege für das Suchergebnis nutzen:

- RSS kann direkt in handelsüblichen Feedreadern angezeigt werden
- ASF wird als Nachfolger von RSS auch bereits von einigen Feedreadern unterstützt
- XHTML kann direkt im Browser ausgegeben werden

Listing 5.5: Anreicherung des Suchergebnisses mit OpenSearch-Elementen

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <feed xmlns="http://www.w3.org/2005/Atom"
3      xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">
4      <!-- ... -->
5      <opensearch:totalResults>4230000</opensearch:totalResults
           >
6      <opensearch:startIndex>21</opensearch:startIndex>
7      <opensearch:itemsPerPage>10</opensearch:itemsPerPage>
8      <opensearch:Query role="request" searchTerms="General␣
           Motors␣annual␣report" startPage="1" />
9      <opensearch:Query role="related" searchTerms="automotive␣
           industry␣revenue" />
10     <!-- ... -->
11 </feed>
```

OpenSearch-Responses können durch zusätzliche, OpenSearch-spezifische, Elemente angereichert werden, die mit dem Präfix *opensearch* beginnen. Die Dokumente bleiben dabei kompatibel zum ursprünglichen Ausgabeformat, indem für die OpenSearch-Ergänzungen ein eindeutiger XML-Namespace benutzt wird. Dieses Verfahren hat den Vorteil, dass eine per OpenSearch ausgelieferte Ergebnisliste ohne weiteres mit allen üblichen, standardkonformen Reader- oder Interpreterprogrammen angezeigt werden kann und nicht zwangsläufig noch ein Extraprogramm installiert werden muss. So kann z.B. ein RSS-Reader die von OpenSearch zurückgegebene, RSS-formatierte Ergebnisseite anzeigen. Echte OpenSearch-Clients können zusätzlich dazu die OpenSearch-spezifischen Informationen interpretieren und dadurch mehrwertige Dienste auf den Rückgabeformaten anbieten. So könnte er zum

Beispiel die im OpenSearch-Header codierte Gesamtanzahl der Treffer anzeigen, wozu der normale RSS-Reader nicht in der Lage ist.
Die OpenSearch-spezifischen Zusatzelemente werden mit Hilfe eines eigenen XML-Namespaces gekennzeichnet und angesprochen. In Listing 5.5 sind einige zulässige OpenSearch-Zusatzelemente aufgelistet. Im Element *totalResults* (Zeile 5) wird die absolute Anzahl an Suchtreffern auf die Anfrage ausgegeben. In *startIndex* (Zeile 6) steht die Nummer des ersten auf der Seite dargestellten Treffers. *itemsPerPage* (Zeile 7) gibt Auskunft darüber, wie viele Suchtreffer-Elemente auf der Ergebnisseite aufgelistet sind. *Query* (Zeile 8 und 9) ist ein vielseitig einsetzbares Element. Das im Listing 5.5 in Zeile 8 gezeigte *Query*-Element mit dem Attribut *role=request* listet z.B. die für die Anfrage benutzten Paramter wie Suchbegriffe oder Startseite des Ergebnisses auf. Die Nutzung von *Query* in Zeile 9 zeigt unter dem Attribut *role=related* alternative Suchbegriffe auf. Weitere Möglichkeiten der Nutzung des *Query*-Elements und die Beschreibung anderer Elemente in OpenSearch-Responses findet sich in der Spezifikation unter [Cli07d].

5.2.3 Extensions

Es gibt derzeit fünf Extensions zu OpenSearch, die offiziell auf der OpenSearch-Seite angeboten werden (Stand: August 2007). Dies sind:

- Referrer Extension (vgl. [Cli07b]): Übermittlung einer Kennung des anfragenden Suchclients bei der Anfrage
- Relevance Extension (vgl. [Cli07c]): Informationen zur Relevanz eines Suchtreffers innerhalb des eigenen Systems
- Parameter Extension (vgl. [Cli07a]): Suchanfragen und andere Parameter per XML an das Suchsystem übergeben
- Suggestion Extension (vgl. [CH07]): Unterbreiten von Vorschlägen zur automatischen Vervollständigung einer Suchanfrage
- Geo Extension (vgl. [Tur07]): Berücksichtigen geographischer Informationen bei der Suche

Insbesondere die Referrer-, Relevance- und Suggestions-Extension sind für das in dieser Arbeit entworfene Suchsystem interessant. Jedoch ist die Suggestions-Extension äußerst komplex und war innerhalb des Bearbeitungszeitraumes dieser Arbeit nicht zu realisieren.
Der Sinn der Parameter-Extension war für das Suchsystem dieser Arbeit nicht ersichtlich, daher wurde auch diese Extension zunächst nicht berücksichtigt.

Die Geo-Extension ist die jüngste der Extensions. Mit ihr ist es möglich, sowohl die Suche auf bestimmte geographische Gegenden einzugrenzen als auch in der Ausgabe eine geographische Angabe unterzubringen, in welcher Gegend der Suchtreffer sich befindet oder auf welche Gegend er sich bezieht. Z.B. kann mit Hilfe dieser Extension der Standort eines Pizzaladens angezeigt werden oder bei einem Artikel über die Reeperbahn ein Stadtplan von Hamburg zum Suchergebnis aufgerufen werden. Die Geo-Extension muss für das in dieser Arbeit beschriebene Suchsystem nicht berücksichtigt werden, da der Dokumentenserver nicht mit geographischen Informationen hantiert.

5.2.4 Erweiterung opensearchopus

OpenSearch ist, wie alle anderen dem Autor dieser Arbeit bekannten Internet-Suchmaschinen und -technologien, dokumentorientiert ausgerichtet. Aufgrund der werksbasierten Arbeitsweise der selbst entwickelten Suchfunktion kann die Ergebnisliste per OpenSearch noch nicht vollständig wie gewünscht zurückgegeben werden. Um diesen Unterschied auszugleichen, wurde eine eigene Erweiterung definiert, mit der der Zusammenhang der Suchtreffer angezeigt und damit sogar die Semantik der Ergebnisse berücksichtigt werden kann.

Realisation

Wie alle Extensions in OpenSearch agiert auch die eigene Extension innerhalb eines eigenen XML-Namespaces. Dieser Namespace lautet *xmlns:opensearchopus= "http://doku.b.tu-harburg.de/opensearch/extensions/opus/1.0/"* und muss im Ergebnisdokument eingebunden werden, wenn die Extension genutzt werden soll. Dieser Namensraum gilt, solange die Erweiterung nicht als produktive OpenSearch-Erweiterung vorgeschlagen bzw. aufgenommen wurde. XML-Namespaces haben einen eindeutigen Pfad als Identifier, der als URL aufgebaut ist. Meist liegt hinter diesem URL eine Erläuterung des Zwecks dieses Namespaces, dies ist jedoch nicht Bedingung. Der Extension wurde der Name opus gegeben, da opus als englisches Wort für Werk die Erweiterung gut beschreibt. Dass es zudem noch im Kontext des gleichnamigen Dokumentenserver-Systems entwickelt wurde, ist Zufall, aber natürlich ein willkommener Nebeneffekt.

5.2.5 Dokumentation opensearchopus

Mit der Erweiterung opensearchopus wird ein Element zur Verfügung gestellt, über das bei einem werksbasierten Suchergebnis festgestellt werden kann, in welchem Teil

(d.h. in welcher Einzeldatei) des Werkes der Suchbegriff gefunden wurde.

Listing 5.6: Auszug aus der OpenSerach-Rückgabe mit opensearchopus-Extension

```
1  <feed xmlns="http://www.w3.org/2005/Atom"
2        xmlns:opensearch="http://a9.com/-/spec/opensearch
           /1.1/"
3        xmlns:opensearchopus="http://doku.b.tu-harburg.de/
           opensearch/extensions/opus/1.0/">
4    <!-- ... -->
5    <entry>
6      <title>New York History</title>
7      <!-- This link should be a reference to the opus-title,
           not to a single file -->
8      <link href="http://doku.b.tu-harburg.de/amerihist/nyc.
         html"/>
9      <!-- ... more data for this particular entry ... -->
10     <!-- This link should be a reference to the particular
         file out of the opus that matches the query -->
11     <opensearchopus:file href="http://doku.b.tu-harburg.de/
         amerihist/new_york.pdf">new_york.pdf</
         opensearchopus:file>
12   </entry>
13   <!-- ... -->
14 </feed>
```

Listing 5.6 demonstriert die Einbindung des definierten Namespaces und die Verwendung des zusätzlichen Elements. Mit der Erweiterung wird das *file*-Element eingeführt. Es enthält den Titel oder den Namen der Datei und nennt als Attribut *href* den URL, unter dem das Dokument direkt zugegriffen werden kann. Das Element kann beliebig oft in jedem Suchtreffer vorkommen, da ein Werk mehrere Dateien enthalten kann, die die Suchbegriffe enthalten. Wenn die Erweiterung genutzt wird, muss das Element mindestens einmal je Treffer vorkommen. Diese Erweiterung wird, wie alle OpenSearch-Extensions, unter die Creative Commons Attribution-ShareAlike 2.5 License[1] gestellt.

[1] http://creativecommons.org/licenses/by-sa/2.5/

Kapitel 6

Implementierung

6.1 Allgemeines

Wie in Abschnitt 3.3 dargelegt, wurde zur Realisation der Suchmaschine das Zend-Framework benutzt. Als Programmiersprache war dadurch PHP5 vorgegeben. Die Implementierung erfolgte objektorientiert. Auch aus diesem Grund musste Version 5 des PHP-Intepreters benutzt werden, denn die Objektorientierung steckte in PHP4 noch im Entwicklungsstadium. Auf der TUBdok-Testmaschine läuft PHP in Version 5.2.3. Bei der Entwicklung wurde Version 1.0.0RC2 des Zend-Frameworks benutzt.

Installation von Zend Zur Installation von Zend-Framework muss das Verzeichnis library/Zend in einen PHP-Library-Pfad kopiert werden. Scripte, die auf einem Zend-Paket aufbauen, müssen dieses inkludieren. Daher muss der Pfad zum Zend-Paket in der PHP-Konfigurationsdatei *php.ini* im *include_path* vorhanden sein.

6.1.1 Umsetzung des Indizierungsablauf in Zend-Lucene

Der in Abschnitt 4.1.3 beschriebene Indizierungsablauf wird im Zend-Lucene-Paket von diversen Klassen realisiert. Zunächst erhält ein Objekt der Klasse *Zend_Search_Lucene_Index_Writer* Zugriff auf die Indexdatei(en). Dabei wird ein schreibender Zugriff benötigt. Das eigentliche Speichern des Index übernimmt die Klasse *Zend_Search_Lucene_Storage_Directory_Filesystem*, über die auch der Speicherort festgelegt werden kann. Im Java-Original von Lucene kann entschieden werden, ob sich der Speicherort im Arbeitsspeicher befindet oder in einem Dateisystem auf der Festplatte abgespeichert werden soll (vgl. [GH05], S. 19). In der Zend-Implementierung ist derzeit nur die Speicherung auf Platte verfügbar. Es gibt zwar eine Klasse *Zend_Search_Lucene_Storage_File_Memory*, jedoch wird diese beim derzeitigen Stand der Implementierung noch nicht genutzt.

Die abstrakte Klasse, die die benötigten Methoden und Variablen im Zusammenhang mit der Analyse definiert, ist in der Zend-Implementierung *Zend_Search_Lucene_Analysis_Analyzer*. Die Klasse wird von einer weiteren abstrakten Klasse *Zend_Search_Lucene_Analysis_Analyzer_Common* beerbt, die wiederum in vier Subklassen implementiert wird (vgl. [Zen07][1]). In den Index aufgenommen werden Objekte der Klasse *Zend_Search_Lucene_Document*. Diese Objekte sind dabei in verschiedene Felder unterteilt, die ihrerseits in *Zend_Search_Lucene_Field*-Objekten repräsentiert werden. *Zend_Search_Lucene_Field* implementiert dabei die im Abschnitt 4.1.3 beschriebenen Lucene-Feldtypen.

6.1.2 Singleton-Implementierung unter PHP

Einen Singleton in PHP zu implementieren ist nicht trivial, da die Lebenszeit von Objekten unklar ist. Es ist nicht eindeutig ersichtlich, wann der Destruktor ausgeführt wird und so ist es nicht möglich, verlässliche Aussagen darüber zu treffen, wann das Objekt wirklich instanziiert ist. Wird der Singleton wie in [PHP07b] vorgeschlagen implementiert, so ist es nicht möglich, die Objektinstanz bei parallelen Scriptstarts ordnungsgemäß zurückgeben zu lassen. Dieses Verhalten ist z.B. bei den als Singleton implementierten Events unerwünscht, denn dabei werden eindeutige Objekte benötigt, damit sich die EventListener an einer eindeutigen Instanz registrieren können.

Listing 6.1: Auszug aus Singleton.class.php

```
1  private static $instanceFile = array();
2
3  protected static function getSingleton($className,
       $classfile){
4    self::$instanceFile[$className] = $classfile;
5    if (file_exists(self::$instanceFile[$className]))
6    {
7      $inst = implode('', file(self::$instanceFile[$className
         ]));
8      $instance = unserialize($inst);
9      $instance->alreadyRunning = true;
10   }
11   else
12   {
```

[1] http://framework.zend.com/apidoc/core/Zend_Search_Lucene/Analysis/Zend_Search_Lucene_Analysis_Analyzer_Common.html

```
13      try
14      {
15        $instance = new $className;
16      }
17      catch (Exception $e)
18      {
19        throw $e;
20        return $instance;
21      }
22      if($instance instanceof Singleton)
23      {
24        $inst = fopen(self::$instanceFile[$className], "w");
25        fwrite($inst, serialize($instance));
26        fclose($inst);
27        echo "Instanz von ".$className." angelegt\n";
28      }
29      else
30      {
31        throw SingletonException("Class '$className' do not
              inherit from Singleton!");
32      }
33    }
34    return $instance;
35  }
36
37  protected function saveInstance($className) {
38    $inst = fopen(self::$instanceFile[$className], "w");
39    fwrite($inst, serialize($this));
40    fclose($inst);
41  } // end of member function saveInstance
42
43  protected function __construct() {     }
```

Die in Listing 6.1 gezeigte Klasse ist eine modifizierte Variante von [com06]. Nach der in Abschnitt 4.2.1 beschriebenen Singleton-Art ist der Konstruktor der Singleton-Klasse als *protected* deklariert (s. Listing 6.1, Zeile 43). Dadurch wird sichergestellt, dass nicht irgendeine Klasse das Objekt erzeugen kann. Der Zugriff auf das Singleton-Objekt geht über die Methode *getSingleton()* (Zeile 3-37). Auch diese Methode ist *protected*, da sie ebenfalls nur in den Singleton implementierenden Klassen aufge-

rufen werden darf. Die implementierenden Klassen, wie z.B. EnabledDocument in Abb. 4.4, greifen auf *getSingleton()* zu und stellen den Aufruf in der Methode *getInstance()* zur öffentlichen Verfügung (s. Listing 6.2).

Listing 6.2: Auszug aus Indexer.class.php

```
1 private static $instanceFile = "/tmp/Indexer.instance";
2
3 //public getter for singleton instance
4 public static function getInstance(){
5   return Singleton::getSingleton(get_class(), self::
      $instanceFile);
6 }
```

Um den nicht eindeutigen Lebenszyklus der Objekte auszugleichen, wurde für die Singleton-Implementierungen eine Zwischenspeicherung in Dateien implementiert. Wie in Listing 6.1 zu erkennen ist, hat das Singleton-Objekt ein Attribut *instanceFile*, in dem der Pfad zur Zwischenspeicherdatei festgelegt wird (Zeile 1). Da es sich um eine Klasse handelt, die sämtliche konkreten Singleton-Implementierungen aufnehmen soll, wurde das Attribut *instanceFile* als Array definiert. Die Zwischenspeicherdatei enthält die aktuelle Objektinstanz in serialisierter Form. Durch die Serialisierung wird das Objekt in eine speicherbare Form umgewandelt.
Wird per *getSingleton()* auf das Singleton-Objekt zugegriffen, wird das Objekt beim ersten Zugriff initialisiert und die Datei angelegt (Zeile 24-26). Die nächsten Zugriffe bekommen als Arbeitsobjekt das aus der Datei entnommene aktuelle Objekt zugewiesen (Zeile 7-8). Dies ist realisiert durch eine Überprüfung, ob die Datei bereits existiert (Zeile 5). Durch das Attribut *alreadyRunning*, das in Zeile 9 gesetzt wird, wird markiert, dass der Indexer bereits instanziiert war und demzufolge im Augenblick des Aufrufs bereits beschäftigt ist. Zusätzlich wird überprüft, ob die *getSingleton()* aufrufende Klasse auch wirklich die Singleton-Klasse implementiert (Zeile 22). Ist dem nicht so, wird eine Exception geworfen (Zeile 31). Nach Abschluss der genannten Überprüfungen liegt, sofern nicht eine Exception ausgelöst wurde, ein Objekt der implementierenden Singleton-Klasse vor, das in Zeile 34 zurückgegeben wird.
Wird das Objekt im Laufe seiner Lebenszeit modifiziert, so muss das neue aktuelle Objekt serialisiert und in der Objekt-Datei gespeichert werden. Das wird über die Methode *saveInstance()* erledigt, die nach einer Änderung des Objektes aufgerufen werden muss. Dieser Ablauf ist in den Zeilen 37-41 implementiert. Der Zugriff auf *saveInstance()* ist dabei wieder aufgrund der Deklaraion als *protected* nur aus der implementierenden Klasse möglich, damit nicht jede Fremdklasse die Möglichkeit

hat, das instanziierte Objekt in der Speicherdatei zu manipulieren.

Da PHP keine Mehrfachvererbung erlaubt, konnten nicht alle als Singleton implementierten Klassen die Singleton-Klasse erweitern. Die Klasse *Indexer* zum Beispiel ist eine Klasse, die von der Klasse *IndexRunner* beerbt wird. Daher kann *Indexer* selbst keine Klasse beerben.
Es gibt noch Probleme bei der Singleton-Implementierung. Wenn es zum Beispiel passiert, dass das Indizierungsprogramm unerwartet endet, so bleibt die Instanzdatei des Indexers trotzdem liegen. Somit wird bei einem weiteren Aufruf der von *Indexer::getInstance()* eine gültige Instanz zurückgegeben, die den Flag *alreadyRunning* gesetzt hat. Die aufrufende Klasse muss also davon ausgehen, dass der Indexer existiert und derzeit beschäftigt ist und schreibt den Auftrag in die Queue. Diese wird allerdings nicht mehr abgearbeitet, weil der Indexer in Wirklichkeit bereits beendet ist und die Queue daher nicht überprüft wird. Dieses Problem wird erst behoben, wenn die Instanzdatei des Indexers manuell gelöscht und der Indexer manuell neu gestartet wird.

6.1.3 Implementierung des Beobachter-Musters und des Befehlsmusters unter PHP

PHP bietet per se keine Event-Handling-Unterstützung, wie sie im Rahmen des vorliegenden Modells benötigt wird (vgl. Abschnitt 4.2.2). Daher wurde eine Event-Handling-Routine selbst konstruiert.
Im Unterschied zum klassischen Event-Handling-Modell werden in der vorliegenden Implementierung Objekte nicht auf ihren Status überprüft, sondern ein Ereignis wird per Methodenaufruf ausgelöst. Dies wäre nichts besonderes, wenn nicht die Event-Objekte beim Aufruf der Event-Methode alle Listener benachrichtigen würden, die an ihnen registriert sind. Beim Aufruf der Event-Methode wird ein *Action*-Objekt gebildet, das direkt an alle Beobachter verteilt wird. Im Fall von *EnabledDocument* wird zum Beispiel die Methode *enableDocument()* ausgelöst, die ein Objekt der Klasse *Action* bildet, das das Kommando *add* und als Parameter die ID des freigeschalteten Dokuments enthält. Ein Beispiel für die Auslösung eines solchen Events bei Freischaltung des Dokuments Nummer 37 wird in Listing 6.3 präsentiert.

Listing 6.3: Auslösen eines Freischaltungsevents aus TUBdok heraus

```
1 $newdoc = EnabledDocument::getInstance();
2 $newdoc->enableDocument("37");
```

Listing 6.4: Auszug aus dem Konstruktor der Klasse Indexer

```
1 $EnabledDocument = EnabledDocument::getInstance();
2 $DeletedDocument = DeletedDocument::getInstance();
3 $EnabledDocument->register(new IndexEventListener("
    NewDocumentWatcher"));
4 $DeletedDocument->register(new IndexEventListener("
    DeleteWatcher"));
```

Jeder das Objekt beobachtende EventListener wird über die Methode *register(Event-Listener)* als Beobachter eingetragen. Im Fall des Indexers wird dies beim Erzeugen der Indexer-Instanz erledigt, wie in Listing 6.4 dargestellt. Die Implementierung der Registrierung und der Benachrichtigung der Listener wird in Listing 6.5 gezeigt.

Listing 6.5: Methode zum Registrieren von Event-Listenern aus EnabledDocument.class.php

```
1  public function register($listenerObject)
2  {
3    if (!in_array(serialize($listenerObject), $this->
       registeredListeners))
4    {
5      echo "Added ".get_class($listenerObject)." as Listener
         ...\n";
6      array_push($this->registeredListeners, serialize(
         $listenerObject));
7      $this->saveInstance();
8    }
9    else
10   {
11     echo get_class($listenerObject)." is already listening
         ...\n";
12   }
13 }
14
15 public function enableDocument($id)
16 {
17   $action = new Action("add", $id);
18   $this->benachrichtige($action);
19 }
20
```

```
21 public function benachrichtige($action)
22 {
23   foreach ($this->registeredListeners as $listener)
24   {
25     $realListener = unserialize($listener);
26     $realListener->doAction($action);
27   }
28 }
```

In Zeile 1-13 im Listing 6.5 ist die Methode zum Registrieren der Listener-Objekte aufgeführt. Das Event-Objekt speichert die horchenden Listener in einem Array. Ist ein Listener bereits unter den registrierten Beobachtern, so wird er nicht nochmal eingefügt (if-Klausel in Zeile 3). Beim Auslösen der Methode *enableDocument(id)* wird das *Action*-Objekt gebildet (Zeile 17) und diese Aktion über die Methode *benachrichtige(aktion)* an alle Listener verteilt (Zeile 18). In der Methode *benachrichtige(aktion)* wird der interne Array mit den Listener-Objekten durchlaufen (Zeile 23-27) und auf allen Listenern die Methode *doAction(aktion)* aufgerufen (Zeile 26). Dabei liegen die Listener-Objekte in serialisiertem Zustand vor, da sie nur auf diese Weise abgespeichert werden können (vgl. Abschnitt 6.1.2). Damit die Aktion auf ihnen ausgeführt werden kann, müssen sie erst deserialisiert werden (Zeile 25).

6.1.4 Implementierung des Proxy-Musters

Die Klasse *IndexRunner* ist eine Proxy-Klasse zum Zugriff auf den Indexer. *IndexRunner* nimmt an den Indexer gerichtete Kommandos entgegen und entscheidet, ob das ihm übergebene Kommando sofort ausgeführt wird, oder ob es in die Queue des Indexers eingetragen wird. Er bietet nur eine Methode zur Aktionsausführung, die sich analog zum Event-Handling-Modell *doAction(aktion)* nennt. Die Implementierung ist in Listing 6.6 dargestellt.

Listing 6.6: Auszug aus IndexRunner.class.php

```
1 public function doAction($action) {
2   if ($this->index->alreadyRunning === true)
3   {
4     $this->indexqueue->addToQueue($action);
5   }
6   else
7   {
8     switch ($action->getCommand())
9     {
```

```
10        case "add":
11          $this->index->addToIndex($action->getParam());
12          break;
13        case "delete":
14          $this->index->removeFromIndex($action->getParam());
15          break;
16        case "build":
17        case "rebuild":
18          $this->index->buildIndex();
19          break;
20        default:
21          // Tu nichts, Aktion nicht implementiert
22          break;
23      }
24    }
25  } // end of member function doAction
```

Wenn der Indexer eine Aktion ausführt, wird von der Singleton-Klasse das Attribut *alreadyRunning* gesetzt (vgl. Listing 6.1, Zeile 9). Dies wird in Listing 6.6, Zeile 2 überprüft. Ist das Attribut gesetzt, so wird das Kommando in die Queue eingetragen (Zeile 4), andernfalls wird es sofort ausgeführt (Zeile 8-23). Der Indexer überprüft nach jedem abgearbeiteten Kommando, ob Einträge in der Queue vorhanden sind und führt diese ggf. aus, bevor er sich beendet.

Der Proxy ist über eine extend-Beziehung mit dem Indexer verknüpft. Dies erlaubt es, die Methoden im Indexer als *protected* zu implementieren, so dass nur Kindklassen auf sie zugreifen können. Da PHP keine Mehrfachvererbung erlaubt, konnte dieselbe Vorgehensweise mit der Queue nicht vorgenommen werden, weil die Queue-Klasse bereits von Singleton abgleitet ist. Daher wurden die Methoden der Queue-Klasse weitgehend als *public* deklariert.

6.1.5 Implementierung des Strategie-Musters und des Fabrikmethode-Musters

Listing 6.7: convert-Methode in der Klasse FileFormat

```
1  public static function convert($doc)
2  {
3    switch ($doc->getMimeType()) {
4      case "text/html":
```

```
5         $html = new HTMLDocument($doc);
6         $volltext = $html->toPlainText();
7         break;
8       case "text/plain":
9         $text = new TextDocument($doc);
10        $volltext = $text->toPlainText();
11        break;
12      case "application/pdf":
13        $pdf = new PDFDocument($doc);
14        $volltext = $pdf->toPlainText();
15        break;
16      case "application/postscript":
17        $ps = new PSDocument($doc);
18        $volltext = $ps->toPlainText();
19        break;
20      default:
21        throw new FileFormatException("Kein Konverter!
            Unbekanntes oder nicht unterstuetztes Dateiformat."
            , "100");
22        return 0;
23    }
24    if ($volltext == "") throw new FileFormatException("Leerer
       Volltext bei ".$doc->_path, "101");
25    return (utf8_encode($volltext));
26  }
```

In den Abschnitten 4.2.5 und 4.2.6 wurden zwei Szenarien dargestellt, bei denen im Kontext der Implementierungen dieser Arbeit Strategien und damit verbunden auch Fabrikmethoden angewendet werden.

Das unter Punkt 1 im Abschnitt 4.2.5 beschriebene Szenraio kümmert sich um die Indizierung verschiedener Dateiformate. Die Klasse *FileFormat* enthält eine einzige Methode. Diese Methode namens *convert(doc)* ist in Listing 6.7 gezeigt. Als Parameter wird dieser Methode die zu konvertierende Datei in Form eines *OPUSDocumentFile*-Objektes übergeben. *OPUSDocumentFile* erbt von der Klasse *OpusFile* eine Methode zur Ermittlung des MIME-Typs der Datei. Über den MIME-Typ ermittelt FileFormat den benötigten Konverter (Zeile 3 des Listings) und bildet als Umsetzung des Fabrikmethode-Musters das passende Objekt (Zeile 4-23). Ist der MIME-Typ mit keinem Konverter verbunden, so wird eine Exception ausgelöst und 0 zurückgegeben (Zeile 21-22). Da es in dieser Methode nur um den Volltext geht,

wird auch im Erfolgsfall nur der aus der Datei extrahierte Volltext zurückgegeben (Zeile 25).
Der Strategieanteil dieser Umsetzung besteht in der Definition eines Interfaces für die einzelnen Konverter-Klassen. Das Interface heißt *FileFormatConverter* und definiert die Methode *toPlainText()*. Die einzelnen Subklassen implementieren dieses Interface und damit diese Methode. In der Implementierung wird festgelegt, wie der Konvertierungsvorgang konkret umgesetzt werden soll. In den meisten Fällen wird sich in der Implementierung eines externen Konvertierungsprogrammes bedient.
Zum Zeitpunkt der Fertigstellung dieser Arbeit sind Klassen zum Konvertieren von HTML, PlainText, PDF und PS vorhanden. Eine Klasse zum Konvertieren von PlainText zu PlainText kann notwendig sein, wenn eine Zeichensatz-Konvertierung vorgenommen werden muss. Daher wurde auch eine solche, auf Anhieb vielleicht nicht sinnvoll erscheinende Klasse bereitgestellt.

6.1.6 Implementierung des Adapter-Musters

Die genauen Bereiche, in denen das Adapter-Muster verwendet wurde, wurden in Abschnitt 4.2.7 besprochen.

SearchEngineAdapter

Die Schnittstelle *SearchEngineAdapter* dient dazu, die Suchanfrage an das Suchsystem weiterzureichen und als Rückgabe eine konforme Ergebnisliste zu erhalten. Sie definiert nur die Methode *find()* als Schnittstellenmethode. Die Implementierung dieser Methode muss mehrerlei leisten:

1. Die Suchanfrage muss an das Suchsystem weitergereicht werden, ggf. muss sie zunächst zerlegt und analysiert werden, um sie kompatibel zum Suchsystem zu machen.

2. Die Rückgabe der Suche (Ergebnisliste) muss in eine QueryHitList konvertiert werden, mit der intern weitergearbeitet wird.

Der Punkt 1 kann beim LuceneAdapter ignoriert werden, da *Zend_Search_Lucene* selbst bereits eine *find()*-Methode implementiert, die den unanalysierten und unzerlegten Suchstring als Parameter erwartet. Die in LuceneAdapter implementierte *find()*-Methode reicht den Suchstring also einfach an *Zend_Search_Lucene::find()* weiter (s. Zeile 11 in Listing 6.8). Punkt 2 ist eine umfangreichere Aufgabe, bei der sich der Adapterklasse *LuceneQueryHitAdpater* bedient wird. Dies passiert in Zeile 19-42 in Listing 6.8. Zunächst wird ein QueryHitList-Objekt angelegt (Zeile 19), zudem werden zwei interne Arrays angelegt (Zeile 20 und 21). Nun wird die von Lucene zurückgegebene Ergebnisliste über foreach durchlaufen (Zeile 24) und

jeder einzelne Treffer verarbeitet. Dies ist ein kompliziertes Unterfangen, weil der aufgebaute Index (wie in Abschnitt 4.1.4 dargelegt) dokumentbasiert aufgebaut ist, die Suche jedoch werksbasiert erfolgen soll.

In der *QueryHitList* steht pro Treffer ein Werk, während aus dem Index pro Treffer ein Dokument zurückgeliefert wird. Jedes Dokument im Index hat eine Referenznummer, die auf das zugeordnete Werk in der OPUS-Datenbank verweist. Um zu überprüfen, ob ein Dokument aus der Lucene-Trefferliste bereits einem Werk in der *QueryHitList* zugeordnet ist, wird der Array *done* benutzt. Die Abprüfung, ob das Werk des gefundenen Treffers bereits im Array *done* enthalten ist, findet in Zeile 28 statt. Ist die Überprüfung negativ, so wird die Werksnummer in *done* aufgenommen (Zeile 30). Außerdem wird der Suchtreffer aus dem Lucene-Format in das interne Format konvertiert (Zeile 31-32) und als *QueryHit*-Objekt in den Ergebnisarray aufgenommen (Zeile 33). Ist das Werk des Dokumentes bereits in *done* enthalten, wird nur das Dokument dem Suchtreffer hinzugefügt (Zeile 39), um anzuzeigen, welches Dokument aus diesem Werk die Kriterien der Suchanfrage konkret erfüllt. Ist jeder Suchtreffer aus der Lucene-Ergebnisliste auf diese Weise verarbeitet, gibt *find()* den intern aufgebauten Rückgabearray zurück (Zeile 46).

Listing 6.8: find()-Methode in LuceneAdapter.class.php

```
1  /**
2   * Suchfunktion: Gibt die Anfrage an das Lucene-System
        weiter
3   * @param string query
4   * @return LuceneQueryHitAdapter
5   * @access public
6   */
7  public function find($query) {
8    try
9    {
10     $index = new Zend_Search_Lucene('/server/htdocs/opus
         -3.0/lib/lucene/luceneindex.entries');
11     $hits = $index->find(utf8_encode(strtolower($query)));
12   }
13   catch (Zend_Search_Lucene_Exception $searchException)
14   {
15     echo "Error: ".$searchException->getMessage()."<br/>";
16   }
17   // Die Suchergebnisse sind jetzt im Lucene-Format
```

```
18    // Die Methode soll aber eine OPUS-konforme QueryHitList
          zurückgeben
19    $hitlist = new QueryHitList();
20    $done = array();
21    $hitlistarray = array();
22    if (count($hits) > 0)
23    {
24      foreach ($hits as $queryHit)
25      {
26        $document = $queryHit->getDocument();
27        $docid = str_replace("nr", "", $document->
              getFieldValue('docid'));
28        if (!in_array($docid, $done))
29        {
30          array_push($done, $docid);
31          $opusHit = new LuceneQueryHitAdapter($queryHit);
32          $curdoc = $opusHit->convertToQueryHit();
33          array_push($hitlistarray, $curdoc);
34        }
35        else
36        {
37          $key = array_search($docid, $done);
38          $opusfile = new OPUSDocumentFile($document->
                getFieldValue('source'), $docid);
39          $hitlistarray[$key]->addFile($opusfile);
40        }
41      }
42    }
43    #$hitlist->add($curdoc);
44    $hitlist->query = $query;
45    $hitlist->queryHits = $hitlistarray;
46    return $hitlist;
47  } // end of member function find
```

QueryHitAdapter

Die Schnittstelle QueryHitAdapter dient dazu, einen im Suchsystem gefundenen Suchtreffer in das intern benutzte Objektformat eines QueryHit zu bringen. Wieder

wird nur eine einzige Methode definiert, um diese Aufgabe zu erfüllen. Die Methode heißt in dieser Schnittstelle *convertToQueryHit()*. Diese Methode ist vollständig in Listing 6.9 anbgedruckt.

Listing 6.9: convertToQueryHit()-Methode in QueryHitAdapter.class.php

```
1  /**
2   * Wandelt einen LuceneQueryHit zu einem intern verwendbaren
        QueryHit
3   * @return QueryHit
4   * @access public
5   */
6  public function convertToQueryHit() {
7    // aus dem Lucene_Search_QueryHit einen QueryHit für OPUS
        machen
8    // Ranking und sonstige Eigenschaften werden aus der
        Lucene-Klasse übernommen
9    $qhit = new QueryHit();
10   $qhit->score = $this->_parent->score;
11   $document = $this->_parent->getDocument();
12   $docid = str_replace("nr", "", $document->getFieldValue('
        docid'));
13   $opusfile = new OPUSDocumentFile($document->getFieldValue(
        'source'), $docid);
14   $opusdoc = new OPUSDocument();
15   $opusdoc->loadRecord($docid);
16   $qhit->_document = $opusdoc;
17   $qhit->addFile($opusfile);
18   return $qhit;
19 } // end of member function getDocument
```

Bei der Konvertierung des Lucene-Suchtreffers zu einem OPUS-Suchtreffer wird die Relevanzbewertung von *Zend_Search_Lucene_Search_QueryHit* übernommen (Zeile 10). Anhand der ID des Werkes und dem Feld *source* aus dem Index wird zunächst die konkrete auf die Suchanfrage matchende Datei ausgelesen (Zeile 11-13). In Zeile 17 wird die Datei in den Suchtreffer übernommen. Auch das Werk, zu dem die Datei gehört, wird dem Suchtreffer zugeordnet. Dies passiert in den Zeilen 14-16: zunächst wird ein Werk-Objekt gebildet (Zeile 14) und der komplette Datensatz geladen (Zeile 15). Nachdem der Suchtreffer auf diese Weise fertig gebaut ist, wird er in Zeile 18 zurückgegeben.

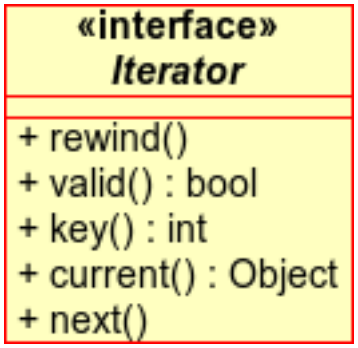

Abbildung 6.1: Iterator-Schnittstelle von PHP

Methode	Parameter	Return	Beschreibung
goto	nummer	void	Springt zu einer gegebenen Indexnummer in der Liste.
getResultList	-	array	Gibt die Originalliste zurück.
getNumberOf Hits	-	int	Ermittelt die Anzahl der Elemente in der Liste.

Tabelle 6.1: Eigene Methoden der Iterator-Implementierung in QueryHitListIterator

6.1.7 Implementierung des Iterator-Musters

PHP5 bietet bereits von Haus aus eine Iterator-Schnittstelle (vgl. [Ber05], S. 45ff), die auch intern bei Methoden zum Durchlaufen von Listen genutzt wird. Das Iterator-Interface von PHP ist in Abb 6.1 dargestellt. Im Fall der Implementierung der Suchfunktion für die Suchmaschine wird das Iterator-Interface benötigt, um die Liste von Suchtreffern zu durchlaufen bzw. weiterzuverarbeiten.
Die Implementierung der Iterator-Schnittstelle ermöglicht es, in PHP-Scripts den foreach-Operator zum Durchlaufen der implementierenden Klasse zu nutzen, was die Verarbeitung der zu iterierenden Liste an einigen Stellen vereinfacht.
Zusätzlich zu den aus dem Interface imlementierten Methoden wurden weitere für den konkreten Anwendungsfall sinnvolle Methoden in der QueryHitListIterator-Klasse untergebracht. Diese sind in Tab. 6.1 beschrieben.

6.2 Ergebnisausgabe

Die Muster Strategie und Fabrikmethode wurden bereits im Zusammenhang mit der Indizierung angewandt (s. Abschnitt 6.1.5). Jedoch unterscheidet sich die Umsetzung der die gleichen Muster verwendenden Ergebnisausgabe von der bereits beschrieben.

Wie schon im Abschnitt 4.2.7 erwähnt, basieren die konkreten Strategie-Klassen beim Suchtreffer-Adapter auf einer abstrakten Klasse. Alle Ausgabeklassen, die die konkreten Strategien in der Musterumsetzung darstellen, sind von der abstrakten OutputFormat-Klasse abgeleitet. In der abstrakten Klasse sind einige für alle Klassen verfügbare Methoden implementiert, aber auch einige abstrakte Methoden definiert, die zur Ausgabe des Formates in den jeweiligen Subklassen implementiert werden müssen. Zu diesen Methoden zählen die beiden öffentlichen Methoden *getOutput()* und *getOpenSearchOutput()*, wie aus der Abb. 4.9b zu entnehmen ist. *getOutput()* liefert das Format der OutputFormat-Klasse ohne weitere Anreicherungen, also reines RSS oder ASF. Bei XHTML liefert *getOutput()* nur einen HTML-formatierten Ausschnitt, der in eine selbst gestaltete Seite eingepasst werden kann. In diesem Ausschnitt sind nur die Suchtreffer aufgelistet. Um eine vollständige XHTML-Seite angezeigt zu bekommen, muss die Methode *getCompleteOutput()* aufgerufen werden.
Die *getOpenSearchOutput()*-Methode liefert das Format mit weiteren, OpenSearch-spezifischen Daten, wie in Abschnitt 5.2.2 beschrieben.
Damit der Client sich nicht selbst um die Erzeugung der OutputFormat-Objekte kümmern muss, steht eine Fabrik-Klasse *OutputFormatFabrik* zur Verfügung. Über diese Klasse kann mit Hilfe der Methode *getObject()* ein Objekt des gewünschten OutputFormats angefordert werden.

6.3 OpenSearch

6.3.1 Allgemein

Im Unterschied zur direkt in den Dokumentenserver eingepassten Suche, die auf die Klasse *Query* zugreift, wird die OpenSearch-Retrieval-Schnittstelle über die Klasse *OpenSearchQuery* gesteuert. So kann implizit durch die Definition eines *OpenSearchQuery*-Objektes festgelegt werden, dass der OpenSearch-Output des jeweiligen Output-Formates benutzt werden soll. Der Output wird in der gleichnamigen Objekteigenschaft gespeichert und kann von dem aufrufenden Script per *getOutput()* abgefragt werden. Der Aufbau der Klasse *OpenSearchQuery* kann dem Diagramm des gesamten Suchsystems im Anhang A entnommen werden.

6.3.2 Extensions

Zwei der im Abschnitt 5.2.3 vorgestellten Erweiterungen von OpenSearch wurden in das System implementiert.

Referrer Extension

Der Client kann einen Identifier-String mit der Suchanfrage abschicken, so dass die OpenSearch-Suchmaschine automatisch bestimmen kann, welches Format dem Client zurückgeliefert werden muss. Dies geschieht in der Methode *analyseReferrer(referrer)* aus der Klasse *OpenSearchQuery*. Die Methode wird bereits im Konstruktor der Klasse aufgerufen. Dadurch kann der Client automatisch erkannt werden und darauf basierend ein passendes Ausgabeformat zurückgegeben werden. Das automatisch ermittelte Ausgabeformat lässt sich vom Nutzer auch selbst setzen, der selbst gesetzte Wert übersteuert dann die automatische Erkennung.
Sinnvoll ist diese Funktion zum Beispiel, wenn die Suchfunktion aus einem RSS-Reader heraus ausgelöst wird, aber nicht RSS als Format angegeben wird. Steht im Referrer, dass es sich um einen RSS-Reader handelt, kann die Methode *analyseReferrer(referrer)* dies erkennen und RSS zurückgeben, auch wenn ansonsten standardmäßig HTML zurückgegeben worden wäre.

Relevance Extension

Listing 6.10: Einfügen der Relevanzbewertung in die Ergebnisrückgabe

```
1 $return .= "<relevance:score>".$hit->score."</relevance:
    score>\n";
```

Die von *Zend_Search_Lucene* für den Suchtreffer ermittelte Relevanzbewertung wird als Element in die OutputFormat-Klassen eingefügt. Dies wird in beiden Fällen über die in Listing 6.10 gezeigte Zeile realisiert, die der Rückgabe eine konform zur Relevance-Extension aufgebaute Zeile hinzufügt.

Kapitel 7

Abschließende Arbeiten

7.1 Integration in TUBdok

Das System wurde zwar in Verknüpfung mit der TUBdok-Datenbasis entwickelt, musste aber im Anschluss an die Implementierung noch in TUBdok integriert werden. An den folgenden Stellen war eine Integration erforderlich:

- Bereitstellung einer Suchseite im Nutzer-Frontend
  `http://linws.b.tu-harburg.de:81/lucene/search.php`
- Bereitstellung der OpenSearch-Schnittstelle im Nutzer-Bereich
  `http://linws.b.tu-harburg.de:81/lucene/opensearch.php`
- Auslösen eines EnabledDocument-Events bei Freischaltung eines Dokuments im Administrationsbereich
  `http://linws.b.tu-harburg.de:81/admin/enableEvent.php`
  Die Datei wird beim Freischaltungsvorgang aufgerufen.
- Auslösen eines DeletedDocument-Events bei Entfernung eines Dokuments im Administrationsbereich
  `http://linws.b.tu-harburg.de:81/admin/deleteEvent.php`
  Die Datei wird automatisch Durchführen eines Löschvorgangs aufgerufen.

Die genannten Adressen liegen auf dem TUBdok-Testsystem und sind, mit Ausnahme des Administrationsbereiches, auf diesem frei zugänglich. Die Quelltexte sind auf der beiliegenden CD einsehbar.

7.2 Finale Tests

Im Laufe der Implementierung wurden schon einige Tests beschrieben. Diese Tests wurden nach Fertigstellung der Implementierung wiederholt und erweitert. Die Test-

abläufe und -ergebnisse werden in diesem Abschnitt beschrieben.

7.2.1 Index-Initialisierung

Zweck des Tests

- Läuft der initiale Indexaufbau ordentlich?
  Allgemeiner Test.
- Werden alle zu indizierenden Dokumente erfasst?
  Allgemeiner Test.
- Kann der Indexer korrekt über die IndexRunner-Klasse angesprochen werden?
  Überprüfung der Proxy-Implementierung.
- Wird das Kommando build bzw. rebuild korrekt gebildet, übergeben und ausgeführt?
  Überprüfung der Befehlmuster-Implementierung.
- Wird je nach Typ des zu indizierenden Dokuments der richtige Konverter gewählt? Überprüfung der Strategie- und Fabrikmuster-Implementierung.

Testablauf Testablauf, erwartetes und tatsächliches Ergebnis sind der Tabelle 7.1 zu entnehmen.

Korrektur

a) Es wurde ein Fehler in der Methode *Indexer::buildEntryIndex()* gefunden und korrigiert. Nach der Behebung lief der Indexer durch und indizierte die Dokumente, denen Volltexte zugeordnet sind, vollständig. Dokumente ohne Volltext wurden nur mit ihren Metadaten erfasst, nicht lesbare Dokumente ebenso. Redundante Einträge konnten vermieden werden.

b) Es war keine Korrektur notwendig.

7.2.2 Index-Update bei Neueinträgen

Zweck des Tests

- Wird das Update des Index bei Neueinträgen ordnungsgemäß durchgeführt?
  Allgemeiner Test.

Nr.	Test	Erwartetes Ergebnis	Resultat	Korrektur
Starten des Initialisierungs-Scripts durch das Script initindex ...				
a	... bei nicht vorhandenem Index	Alle Einträge in der Datenbank werden in den Index aufgenommen, Einträge ohne Volltext werden zumindest als Metadaten indiziert	✗	✓
b	... bei vorhandenem Index	Alle Einträge in der Datenbank werden in den Index aufgenommen, Einträge ohne Volltext werden zumindest als Metadaten indiziert - der vorhandene Index wird dabei überschrieben	✓	-

Tabelle 7.1: Tests zur Index-Initialisierung

- Kann eine eindeutige Instanz des EnabledDocuments angesprochen werden?
  Überprüfung der Funktion der Singleton-Implementierung.
- Wird das Kommando add korrekt gebildet, übergeben und ausgeführt?
  Überprüfung der Funktion des Befehlsmusters.
- Wird das Kommando an die richtigen Listener verteilt?
  Überprüfung der Implementierung des Event-Modells.
- Wird je nach Typ des zu indizierenden Dokuments der richtige Konverter gewählt?
  Überprüfung der Strategie- und Fabrikmuster-Implementierung.

Testablauf Testablauf, erwartetes und tatsächliches Ergebnis sind der Tabelle 7.2 zu entnehmen. Alle Tests verliefen erfolgreich, eine Korrektur war nicht notwendig.

7.2.3 Index-Update bei Entfernung von Einträgen

Zweck des Tests

- Wird der Index beim Löschen von Dokumenten ordnungsgemäß aktualisiert?
  Allgemeiner Test.
- Kann eine eindeutige Instanz des DeletedDocuments angesprochen werden?
  Überprüfung der Funktion der Singleton-Implementierung.

Nr.	Test	Erwartetes Ergebnis	Resultat	Korrektur
Auslösen eines Enabled-Events ...				
a	... bei nicht vorhandenem Index	Fehlermeldung	✓	-
b	... bei vorhandenem Index und Eintrag ist bereits indiziert	Löschen des alten Eintrags und erneute Indizierung des Eintrags	✓	-
c	... bei vorhandenem Index und Eintrag ist noch nicht indiziert	Eintrag wird in den Index aufgenommen und ist anschließend auffindbar	✓	-
b	... Integrationstest: Freischaltung eines neuen Eintrags aus dem normalen Workflow heraus	Neueintrag wird in den Index aufgenommen und ist anschließend auffindbar	✓	-

Tabelle 7.2: Tests zur Aktualisierung des Index nach einem Neueintrag

- Wird das Kommando remove korrekt gebildet, übergeben und ausgeführt? Überprüfung der Funktion des Befehlsmusters.
- Wird das Kommando an die richtigen Listener verteilt? Überprüfung der Implementierung des Event-Modells.

Testablauf Testablauf, erwartetes und tatsächliches Ergebnis sind der Tabelle 7.3 zu entnehmen. Die Tests a bis c verliefen erfolgreich, bei Test d war eine Korrektur erforderlich.

Korrektur

a) Keine Korrektur erforderlich

b) Keine Korrektur erforderlich

c) Keine Korrektur erforderlich

d) Beim Entfernen im Administrationsfrontend wird der Fehler angezeigt, dass das Dokument im Index nicht gefunden werden kann. Der Eintrag wird zwar aus der Opus-Datenbank entfernt, nicht aber aus dem Index. Durch eine Modifikation an den Adapterklassen konnte der Fehler zwar nicht beseitigt, aber ausgeglichen werden. Das Suchergebnis wird anhand seines Datenbankeintrages in *LuceneQueryHitAdapter* vervollständigt. Stellt die Konvertierungsmethode in *LuceneQueryHitAdapter* fest, dass der gefundene Treffer nicht in der

Nr.	Test	Erwartetes Ergebnis	Resultat	Korrektur
Auslösen eines Delete-Events ...				
a	... bei nicht vorhandenem Index	Fehlermeldung	✓	-
b	... bei vorhandenem Index und Eintrag ist nicht vorhanden	Meldung, dass Eintrag nicht vorhanden ist und daher nicht entfernt werden kann	✓	-
c	... bei vorhandenem Index und Eintrag ist vorhanden	Der gelöschte Eintrag ist nach Verarbeitung der Aktion aus dem Index verschwunden	✓	-
d	Integrationstest: Löschen eines Eintrags aus dem normalen Workflow heraus	Der gelöschte Eintrag ist nach Verarbeitung der Aktion aus dem Index verschwunden	✗	✗

Tabelle 7.3: Tests zur Aktualisierung des Index nach einer Löschaktion

Datenbank verzeichnet ist, gibt sie anstatt des *QueryHit*-Objektes den booleschen Wert *false* zurück. Dies erkennt der aufrufende *LuceneAdapter* und nimmt den Treffer nicht in die Ergebnisliste auf. Dieser Workaround verhindert zwar die Ausgabe toter Links, verbessert aber nicht die Indexqualität. Daher muss der Fehler für den produktiven Einsatz der Suchmaschine noch korrigiert werden.

7.2.4 Verhalten bei Parallelisierung

Zweck des Tests

- Arbeitet die Queue korrekt?
  Allgemeiner Test.
- Ist die Queue tatsächlich nur einmal instanziierbar?
  Überprüfung der Singleton-Implementierung.
- Werden die Kommandos aus der Queue korrekt ausgeführt?
  Überprüfung der Implementierung des Befehlsmusters.

Testablauf Der Testablauf ist komplex und kann daher nicht in der zusammenfassenden Tabelle allein dargestellt werden. Das erwartete und tatsächliche Ergebnis

Test	Erwartetes Ergebnis	Resultat	Korrektur
Abarbeitung der Queue	Die parallel gestarteten Abläufe werden in der Queue gespeichert und nach Abschluss der Initialisierung korrekt abgearbeitet	✗	✗

Tabelle 7.4: Queue-Funktionstest

sind der Tabelle zu entnehmen 7.4.
Mit dem Script *test/init_index.php* wird eine Initialisierung des Index gestartet. Während der Indexer läuft, werden mit dem Script *test/indextest* von einer anderen Shell aus drei weitere Dokumente in den Volltextbaum kopiert, um dadurch mehrere parallele Freischaltungsvorgänge zu simulieren. Ebenfalls parallel von einer anderen Shell aus wird per *test/delete_index.php* eines der Dokumente aus dem Volltextbaum wieder aus dem Index entfernt.

Testergebnis Die Einträge aus dem parallel gestarteten Script landen zwar ordnungsgemäß in der Queue und werden auch ordnungsgemäß abgearbeitet, aber nach der Abarbeitung des letzten Eintrages terminiert der Indexer nicht. Der Prozess muss manuell beendet werden. Der Fehler ließ sich in mehreren Testläufen reproduzieren, allerdings immer erst, wenn eine Nachindizierung veranlasst wurde. Unproblematisch war eine Initialisierung und nebenläufig eine Löschaktion.

Korrektur Es treten offensichtlich nur Fehler auf, wenn es um die Nachindizierung von Einträgen geht. Dieses Problem würde im Produktionsbetrieb gar nicht unbedingt bemerkt werden, aber die Integrität des Index dennoch gefährden. Daher muss vor dem Produktiveinsatz des Systems für dieses Problem eine Lösung gefunden werden. Im Rahmen dieser Arbeit konnte es nicht mehr gelöst werden.

7.2.5 Korrekte Verhaltensweise bei der Indizierung nicht lesbarer Formate

Zweck des Tests

- Werden alle Formate wie erwartet in den Index aufgenommen?
  Überprüfung der Umsetzung des Ablaufdiagramms für den Indexer, das in Abb. 4.2 dargestellt wurde.

Nr.	Test	Erwartetes Ergebnis	Resultat	Korrektur
Überprüfen des Index mit Luke anhand der Anfrage ...				
a	... Bernhard AND Sell	Fünf Treffer	✗	✓
b	... author:monien	Ein Treffer	✓	-
c	... author:kreuzer	Ein Treffer	✗	✓

Tabelle 7.5: Tests zur korrekten Indizierung

Testablauf Testablauf, erwartetes Ergebnis und die tatsächlichen Ergebnisse sind in der Tabelle 7.5 zusammengefasst.
Durch die Benutzung des Programmes Luke anstatt der selbst entwickelten Suchoberfläche sollte sichergestellt werden, dass die Ergebnisse nicht auf Fehler in der eigenen Entwicklung zurückzuführen sind, sondern wirklich an der fehlerhaften Indizierung liegen müssen.

Korrektur

a) Es werden nur vier anstatt der fünf erwarteten Dokumente gefunden. Die eigentlich erwartete ID 64 fehlt.
Der Fehler wurde im Konvertierungsvorgang lokalisiert. Die Tokenisierung in Lucene arbeitet grundsätzlich mit Leerzeichen. Durch einen Fehler in der Kontvertermethode wurden Zeilenumbrüche ersatzlos aus dem Originaltext entfernt. Dadurch wurde aus dem Wort *Bernhard* im Volltext das Wort *nbernhard*, weil *Bernhard* am Zeilenanfang stand und die Zeile vorher mit einem *n* endete. Eine Suchanfrage nach *nbernhard* ergab tatsächlich das Dokument 64. Zur Korrektur wurde der zeilenumbruch beim Konvertieren in ein Leerzeichen umgewandelt. Die Wiederholung des Tests verlief erfolgreich.

b) Keine Korrektur notwendig.

c) Es werden zwei Treffer gefunden, das Dokument liegt also doppelt im Index vor.
Zusätzlich zum vollständig indizierten Dokument wurden die Metadaten als separater Datensatz abgespeichert. Im Volltextverzeichnis des Dokuments lag noch eine Signaturdatei, die mit der Dateiendung .asc beabsichtigtermaßen nicht indiziert wurde. Es lag also ein Fehler in der Überprüfung vor, ob für den Eintrag bereits Daten indiziert wurden. Dieser Fehler wurde korrigiert. Die Wiederholung des Tests lief erfolgreich.

Test	Erwartetes Ergebnis	Resultat	Korrektur
Suche nach *Bernhard AND Sell* über die selbst erstellte Eingabemaske und über Luke	Vergleichbare Ergebnismengen	✓	-

Tabelle 7.6: Test der Adapterklassen

7.2.6 Übereinstimmung bei Recherchen über die eigene Suchoberfläche und direkter Suche mit Luke

Zweck des Tests

- Sind die Treffer bei Anfragen über die eigene Suchoberfläche identisch mit denen, die bei einer direkten Recherche über das Java-Programm Luke erzielt werden?
  Überprüfung der Funktion der Umsetzung des Adapter-Musters.

Testablauf Testablauf und das Ergebnis sind der Tabelle 7.6 zu entnehmen. Zum erwarteten Ergebnis noch einige Erläuterungen:
Die Ergebnismengen mussten nicht auf Anhieb identisch sein. Bei der Suchanfrage in Luke wird die Treffermenge dokumentorientiert ermittelt, bei der eigenen Suche werksorientiert. Daher musste die Luke-Ergebnisliste manuell auf eine werksbasierte Sortierung gebracht werden, indem die mehrfach gefundenen docids nur als ein Treffer gezählt wurden. Die so ermittelte Trefferanzahl musste mit der aus der selbst erstellten Suchmaske übereinstimmen.

7.2.7 Unterstützung verschiedener Suchstrategien und -methoden bei der Recherche über die eigene Suchmaske

Zweck des Tests

- Werden alle Suchmethoden korrekt unterstützt?
  Überprüfung der in Abschnitt 4.1.7 beschriebenen Suchmethoden.

Testablauf Testablauf und erwartetes Ergebnis sind der Tabelle 7.7 zu entnehmen.

Testergebnis Die in der Tabelle 7.7 dargestellten Resultate bedürfen einer weiteren Erläuterung.

a) Die Suche mit Wildcards ergibt keine Treffer. Eine entsprechende Fehlermeldung, in der erklärt wird, dass Wildcards nicht unterstützt werden, erscheint. Da die Trunkierung als optionales Suchelement deklariert wurde, war eine Korrektur nicht erforderlich.

b) Nach Durchführung der Suche sollte der Suchbegriff wieder im Eingabefeld stehen. Das ist normalerweise auch der Fall, jedoch tritt ein Fehler bei der Suche mit Anführungszeichen auf. Hier erscheint im Eingabefeld nur ein Backslash. Die Suche wird durchgeführt, aber die Ergebnisliste erweckt den Eindruck, dass eine normale ODER-Suche stattgefunden hat anstatt der gewünschten Phrasensuche. Zum Test wurde im Anschluss nach *Technische AND Universität* gesucht. Die Treffermenge ist in der Tat unterschiedlich. Die Phrasensuche ist demzufolge noch nicht korrekt implementiert. Da sie abs optional deklariert wurde, war eine Korrektur im Rahmen dieser Arbeit jedoch noch nicht notwendig.

c) Die Anfrage nach *author:marahrens* ergibt einige Treffer und somit ist belegt, dass die Feldsuche grundsätzlich funktioniert. Der Sachverhalt der fehlerhaften Suche nach Nummern in der Zend-Implementierung wurde bereits in Abschnitt 4.1.6 ausführlich dargestellt.

d) Suchanfrage 1 ergibt genau einen Treffer, Suchanfrage 2 elf und Suchanfrage 3 sieben. Als Vergleichswert wurden auch beide Suchbegriffe jeweils einzeln gesucht. XML ergibt dabei acht Treffer und query 4 Treffer. Daraus lässt sich schließen, dass die Suche mit boole'schen Operatoren funktioniert.

e) Die Suchanfrage ergibt 26 Treffer, lässt man die Klammern weg sind es 27. Der Unterschied in der Ergebnismenge lässt darauf schließen, dass die Klammerung korrekt funktioniert.

7.2.8 Suche per Eingabemaske auf dem System

Zweck des Tests

Nr.	Test	Erwartetes Ergebnis	Resultat	Korrektur
a	Trunkierung: Suchanfrage nach *XML?* und nach *XML**	Plausible Ergebnisliste	✗	-
b	Phrasensuche: Suchanfrage nach *"Technische Universität"*	Plausible Ergebnisliste	✗	-
c	Feldsuche: Suchanfrage nach *docid:nr82* und *author:marahrens*	Plausible Ergebnisliste	✗ ✓	✗
d	Suche mit Boole'schen Operatoren: Suchanfrage nach *XML and Query* und nach *XML or query* und nach *xml and not query*	Plausible Ergebnisliste	✓	-
e	Klammerung bei der Suche: Suchanfrage nach *(informationskompetenz or bibliothek) and hamburg*	Plausible Ergebnisliste	✓	-

Tabelle 7.7: Tests verschiedener Suchmethoden

Test	Erwartetes Ergebnis	Resultat	Korrektur
Suche nach *Bernhard AND Sell* über die selbst erstellte Eingabemaske	Die Ergebnisliste entspricht dem Format und es werden alle Treffer hintereinander ohne Fehlermeldungen ausgegeben.	✓	-

Tabelle 7.8: Test zur Durchführung der Suche

- Wird die Suche und die HTML-Ausgabe beim Suchen über die Eingabemaske direkt im System korrekt ausgeführt?
  Überprüfung der Funktion des Iterators.

Testablauf Testablauf, erwartetes und tatsächliches Ergebnis sind der Tabelle 7.8 zu entnehmen. Der Test verlief erfolgreich.

7.2.9 Suche per Retrieval-Schnittstelle

Zweck des Tests

- Lässt sich die Retrieval-Schnittstelle korrekt ansprechen?
  Allgemeiner Test.
- Ist die Rückgabe syntaktisch korrekt?
  Überprüfung der Funktion der Implementierung des Strategie-Musters und damit verbunden des Fabrikmethode-Musters.

Testablauf Absetzen einer Suchanfrage auf die opensearch.php-Datei

über den Webbrowser Mozilla Firefox mit HTML-Rückgabe
`http://linws.b.tu-harburg.de:81/lucene/opensearch.php`
Parameter:
q=xml

über den Feedreader BlogBridge mit RSS-Rückgabe
`http://linws.b.tu-harburg.de:81/lucene/opensearch.php`
Parameter:
q=(informationskompetenz%20or%20bibliothek)%20and%20hamburg
format=rss

über den Feedreader BlogBridge mit Atom-Rückgabe
`http://linws.b.tu-harburg.de:81/lucene/opensearch.php`

Nr.	Test	Erwartetes Ergebnis	Resultat	Korrektur
a	Ergebnisanzeige HTML	Korrekte HTML-Ausgabe	✓	-
b	Ergebnisanzeige RSS in Feedreader	Korrekte RSS-Ausgabe	✓	-
c	Ergebnisanzeige Atom in Feedreader	Korrekte Atom-Ausgabe	✗	✗

Tabelle 7.9: Tests zur Ergebnis-Ausgabe per Retrieval-Schnittstelle

Parameter:
q=(informationskompetenz%20or%20bibliothek)%20and%20hamburg
format=atom

Testergebnis Das erwartete Testergebnis ist in Tabelle 7.9 dargestellt. Test a und Test b liefen erfolgreich, Test c ergab eine kleine Unstimmigkeit: in BlogBridge wurde die Dokumentzusammenfassung nicht korrekt angezeigt. Eine Überprüfung mit dem Atom-Format im Browser ergab keine syntaktischen Probleme, es lag also ein Interpretationsproblem vor. Eine Behebung und nähere Analyse des Fehlers wird zu einem späteren Zeitpunkt vorgenommen.

7.3 Anschluss an externe Dienste

Der Anschluss an externe Dienste ist in Zeiten des Web 2.0 besonders beliebt. Die in dieser Arbeit entwickelte Suchmaschine kann durch die OpenSearch-Schnittstelle an externe Dienste angeschlossen und von dort aus genutzt werden.
Drei Beispiel für externe Dienste sind hier aufgeführt.

- A9
  A9 kann aufgrund des Outputformats OpenSearch auf Dokumentenserverseite als Metasuchsystem agieren.[1]

- MetaGer
  Von MetaGer-Seite wurde eine vorläufige Einbindung der OpenSearch-

[1]Ein Test des erzeugten OpenSearch-Formates kann auch ohne eine verbindliche Anmeldung bei A9 durchgeführt werden. Ein Testformular findet sich unter `http://opensearch.a9.com/-/opensearch/`. Es muss die Adresse der OpenSearch Description-Datei des Dienstes angegeben werden. Der Test verlief erfolgreich.

Schnittstelle bereits vorgenommen. Die Tests auf dieser Seite verliefen jedoch noch nicht gänzlich zufriedenstellend.

- Metasuchmaschine für OPUS-Server
  Die gemeinsame Suche über OPUS-Systeme kann auch auf der OpenSearch-Schnittstelle aufsetzen und dadurch verbessert werden. Derzeit fragt die gemeinsame OPUS-Suche[1] die Metadatensuche der einzelnen OPUS-Systeme ab. Die Volltexte werden nicht berücksichtigt. Mit der OpenSearch-Suche könnten die Volltexte gleich mitdurchsucht werden und durch das Layout-unabhängige Rückgabeformat könnte die Ergebnisseite ansprechender sortiert und angeordnet werden.

[1] `http://elib.uni-stuttgart.de/opus/gemeinsame_suche.php`

Kapitel 8

Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit konnte wie beschrieben eine Suchfunktion für OPUS entwickelt werden. Die Suchfunktion zeichnet sich dadurch aus, dass sie werksbasiert arbeitet und damit nicht, wie bei Internet-Suchmaschinen sonst üblich, dokumentbasiert. Diese Vorgehensweise kann vielleicht helfen, Suchergebnisse übersichtlicher zu präsentieren, da sie in Kontexte zusammengefasst werden können, was bei der allein dokumentenbasierten Suche nicht ohne weiteres möglich ist.
Wie jede Software haben auch die in der vorliegenden Arbeit entwickelten Systeme Fehler, die im Rahmen der Bearbeitungszeit nicht behoben werden konnten.

- Der in manchen Fällen nicht terminierende Indexer.
- Die nicht funktionierende Entfernung eines Eintrags aus dem Index bei Löschung einer Publikation.
- Die nicht funktionierende Suche nach Nummern in *Zend_Search_Lucene*.
- Der in BlogBridge nicht genannte Abstract im Atom-Rückgabeformat bei OpenSearch.

Auch einige Erweiterungsmöglichkeiten für die Suchmaschine können bereits genannt werden:

- Verbesserung der Ranking-Berechnung bzw. Anpassung der Berechnung an die werksbasierte Vorgehensweise.
- Unterstützung zusätzlicher Suchmethoden wie Trunkierung und Phrasensuche.
- Anbieten der Suggestions-Extension von OpenSearch.
- Komplettierung der Einbindung in MetaGer.

- Anmeldung bei A9.
- Ersetzen der auf Seite 17 beschriebenen Suche nach Metadaten in OPUS durch die neu geschaffene Suchfunktion.

Die entwickelte Suchfunktion ist auf TUBdok ausgelegt und muss angepasst werden, wenn sie auch auf anderen OPUS-basierten Servern verwendet werden soll. Dies war zu erwarten, da TUBdok einige Besonderheiten aufweist, die beim derzeitigen Entwicklungsstand nicht ohne weiteres auf andere OPUS-basierte Repositories übertragen werden können. Zudem ist die Entwicklung von OPUS in starkem Fluss. Im Rahmen der Weiterentwicklung von OPUS werden derzeit Planungen für Version 4.0 angestellt. Für diese Version soll unter anderem das Datenmodell überarbeitet werden. Da das Datenmodell zum Zeitpunkt der Fertigstellung dieser Arbeit noch nicht feststeht, konnte es für die vorliegende Arbeit noch nicht berücksichtigt werden. Auch aus diesem Grund war es nicht möglich, die Suchfunktion schon jetzt universell an alle OPUS-Server anzupassen. Auch eine Suchmaschine soll in Version 4.0 von OPUS integriert werden. Da der Autor dieser Arbeit in die OPUS-Entwicklergruppe aktiv mitarbeitet, können die Ergebnisse dieser Arbeit als Grundlage für diese Suchmaschine dienen.

Die eigene Entwicklung der Extension opensearchopus kann dem OpenSearch-Team vorgestellt und zur Aufnahme in den OpenSearch-Standard vorgeschlagen werden.

Insgesamt ist mit dieser Arbeit eine funktionierende Suchmaschine für TUBdok/ OPUS entstanden, die durch die Einbeziehung von Entwurfsmustern flexibel gehandhabt werden kann.
Mit der werksbasierten Suche wurde ein vorher noch nicht realisiertes Konzept eingeführt, das für Dokumentenserver und andere Dienste, die eine Unterteilung ihres Datenbestandes in Werke vornehmen können, möglicherweise interessant ist. Dass es anwendbar ist, wurde in der vorliegenden Arbeit gezeigt.

Literaturverzeichnis

[as05] AS. *Deutsches Suchmaschinen-Forum mahnt Gefahr durch Monopole an. Debatte über Regulierung im europäischen Maßstab.* Mai 2005. Online im Internet unter URL `http://de.internet.com/index.php?id=2035544`. Abgerufen am 18.08.2007

[Ber05] BERGMANN, Sebastian: *Professionelle Softwareentwicklung mit PHP5. Objektorientierung - Entwurfsmuster - Modellierung - Fortgeschrittene Datenbankprogrammierung.* 1. Aufl. Heidelberg : dpunkt.verlag, 2005

[CH07] CLINTON, DeWitt ; HUGHES, Joe: *OpenSearch Suggestions extension*, 2007. Online im Internet unter URL `http://www.opensearch.org/Specifications/OpenSearch/Extensions/Suggestions/1.0`. Abgerufen am 01.07.2007

[Cli07a] CLINTON, DeWitt: *OpenSearch Parameter extension*, 2007. Online im Internet unter URL `http://www.opensearch.org/Specifications/OpenSearch/Extensions/Parameter/1.0`. Abgerufen am 01.07.2007

[Cli07b] CLINTON, DeWitt: *OpenSearch Referrer extension*, 2007. Online im Internet unter URL `http://www.opensearch.org/Specifications/OpenSearch/Extensions/Referrer/1.0`. Abgerufen am 01.07.2007

[Cli07c] CLINTON, DeWitt: *OpenSearch Relevance extension*, 2007. Online im Internet unter URL `http://www.opensearch.org/Specifications/OpenSearch/Extensions/Relevance/1.0`. Abgerufen am 01.07.2007

[Cli07d] CLINTON, DeWitt: *Spezifikation OpenSearch 1.1 Draft 3*, 2007. Online im Internet unter URL `http://www.opensearch.org/Specifications/OpenSearch/1.1`. Abgerufen am 01.07.2007

[Col07] COLE, Jim. *Re: htdig 3.2b6 compiling problem.* 2007. Online im Internet unter URL `http://www.nabble.com/Re%3A-htdig-3.2b6-compiling-problem-p8189785.html`. Abgerufen am 06.06.2007

[com06] EYVINDH79 AT GMAIL DOT COM. *Kommentar zu PHP: Pattern*. 2006. Online im Internet unter URL `http://www.php.net/manual/de/language.oop5.patterns.php#71579`. Abgerufen am 17.08.2007

[DFG07] DFG. *DFG - Deutsche Forschungsgemeinschaft*. 2007. Online im Internet unter URL `http://www.dfg.de`. Abgerufen am 25.08.2007

[DIN07] DINI. *DINI-zertifizierte Server*. 2007. Online im Internet unter URL `http://www.dini.de/no_cache/service/dini-zertifikat/zertifizierte-server/`. Abgerufen am 19.08.2007

[Eic06] EICHSTÄDT, Thimo: *Evaluierung von Suchmaschinensoftware*, Regionales Rechenzentrum für Niedersachsen, Diplomarbeit, 2006. – Diplomarbeit

[FB07a] FRANZ, Markus ; BLANCKE, Thorsten. *Metager2 : Technologie*. 2007. Online im Internet unter URL `http://metager2.de/technology.php`. Abgerufen am 18.08.2007

[FB07b] FRANZ, Markus ; BLANCKE, Thorsten. *Metager2 : Vorteile*. 2007. Online im Internet unter URL `http://metager2.de/advantages.php`. Abgerufen am 18.08.2007

[FFSB04] FREEMAN, Eric ; FREEMAN, Elisabeth ; SIERRA, Kathy ; BATES, Bert: *Head First Design Patterns*. O'Reilly, 2004

[GH05] GOSPODNETIĆ, Otis ; HATCHER, Erik: *Lucene in Action*. Manning, 2005

[GHJV01] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, 2001

[ht:05] The ht://Dig Group: *How do I index PDF files?* 2005. Online im Internet unter URL `http://www.htdig.org/FAQ.html#q4.9`. Abgerufen am 22.08.2007

[htd02] *ht://dig Configuration file format – Attributes*, 2002. Online im Internet unter URL `http://www.htdig.org/attrs.html`. Abgerufen am 07.06.2007

[htd05] *Htdig - Internet in Binliotheken*, 2005. Online im Internet unter URL `http://www.internet-in-bibliotheken.de/index.php/HtDig`. Abgerufen am 07.06.2007

[Int05] The Internet Engineering Task Force IETF: *RFC 4287*. 2005. Online im Internet unter URL `http://www.ietf.org/rfc/rfc4287.txt`. Abgerufen am 29.06.2007

[iPr06] iPROSPECT. *iProspect Search Engine User Behavior Study*. April 2006. Online im Internet unter URL `http://www.iprospect.com/about/whitepaper_seuserbehavior_apr06.htm`. Abgerufen am 19.08.2007

[Jea06] JEANNENEY, Jean-Noël: *Googles Herausforderung. Für eine europäische Bibliothek*. 1. Berlin : Wagenbach, Februar 2006

[Kar03] KARZAUNINKAT, Stefan: *Die Suchfibel*. 2. Aufl. Klett, 2003

[Kar04] KARZAUNINKAT, Stefan: *Beziehungsgeflecht der Suchdienste in Deutschland und international*. Dezember 2004. – Stand: 17.12.2004. Online im Internet unter URL `http://www.suchfibel.de/5technik/images/suchmaschinereien_gross.gif`. Abgerufen am 18.08.2007

[Kof01] KOFLER, Michael: *MySQL. Einführung, Programmierung, Referenz*. München : Addison-Wesley, 2001

[Lew05] LEWANDOWSKI, Dirk: *Web information retrieval*. Frankfurt am Main : Dt. Ges. für Informationswiss. und Informationspraxis, 2005. – ISBN 3–925474–55–2. Online im Internet unter URL `http://www.durchdenken.de/lewandowski/web-ir`. Abgerufen am 01.05.2007

[Mar05] MARAHRENS, Oliver: Prüfung und Erweiterung der technischen Grundlagen des Dokumentenservers OPUS zur Zertifizierung gegenüber der DINI anhand der Installation an der TU Hamburg-Harburg / Universitätsbibliothek der TU Hamburg-Harburg. 2005. – Forschungsbericht. urn:nbn:de:gbv:830-opus-827. Online im Internet unter URL `http://doku.b.tu-harburg.de/volltexte/2005/82/pdf/projektbericht.pdf`. Abgerufen am 01.05.2007

[Mar07] MARAHRENS, Oliver: Europa und die Informationsgesellschaft. Ein Bericht zum SuMa-eV-Forum am 28. September 2006 in Berlin. In: *ZfBB Zeitschrift für Bibliothekswesen und Bibliographie* 1 (2007), S. 35–40

[Neb04] NEBEL, Michael: Technische Realisierung von Mini-Suchern. In: *1. SuMa-eV Forum*, 2004, Online im Internet unter URL `http://www.nebel.de/projekte/Vortrag-20041122/index.html`. Abgerufen am 06.06.2007

[OAI04] Lagoze, Carl (Hrsg.) ; de Sompel, Herbert V. (Hrsg.) ; Nelson, Michael (Hrsg.) ; Warner, Simeon (Hrsg.). *The Open Archives Initiative Protocol for Metadata Harvesting*. 2004. Online im Internet unter URL `http://www.openarchives.org/OAI/openarchivesprotocol.html`. Abgerufen am 09.06.2007

[OAI07] OAI. *Open Archives Initiative. Standards for Repository Interoperability*. 2007. Online im Internet unter URL `http://www.openarchives.org/`. Abgerufen am 19.08.2007

[Ope07] *OpenSearch search clients*. 2007. Online im Internet unter URL `http://www.opensearch.org/Community/OpenSearch_search_clients`. Abgerufen am 26.08.2007

[OPU06] Stuttgart, Universität (Hrsg.). *OPUS - Beschreibung und Grundsätze*. 2006. Online im Internet unter URL `http://elib.uni-stuttgart.de/opus/doku/about.php`. Abgerufen am 09.06.2007

[PHP07a] *PHP 4 end of life announcement*. Juli 2007. Online im Internet unter URL `http://www.php.net/index.php#2007-07-13-1`. Abgerufen am 22.08.2007

[PHP07b] *PHP: Pattern*. online. 2007. Online im Internet unter URL `http://www.php.net/manual/de/language.oop5.patterns.php`. Abgerufen am 04.08.2007

[Ren07] Renner, Jens: *Don't be evil - Wie wir Google nutzen und wie Google uns nutzt*. 2007. – Präsentation auf einem Workshop zum Thema Information Retrieval an der TFH Wildau am 26.03.2007. Online im Internet unter URL `http://www.tfh-wildau.de/tfhbib/media/Google_Renner.pdf`. Abgerufen am 01.05.2007

[RSS07] *RSS 2.0 Specification*. 2007. Online im Internet unter URL `http://www.rssboard.org/rss-2-0`. Abgerufen am 19.08.2007

[SB98] Sander-Beuermann, Wolfgang. *Meta-Strukturen und -Algorithmen in Internet-Suchmaschinen*. November 1998. Online im Internet unter URL `http://meta.rrzn.uni-hannover.de/meta-strukt/`. Abgerufen am 19.08.2007

[SB00a] Sander-Beuermann, Wolfgang. *Im Westen nichts Neues? - Möglichkeiten und Grenzen im explodierenden Cyberspace*. September 2000. On-

line im Internet unter URL http://metager.de/dgd2000/. Abgerufen am 19.08.2007

[SB00b] Sander-Beuermann, Wolfgang: Neues und Megatrends bei Suchmaschinen. In: *InetBib Tagung 2000*, 2000, Online im Internet unter URL http://www.metager.de/inetbib2000/. Abgerufen am 19.08.2007

[SB07] Sander-Beuermann, Wolfgang. *[InetBib] Warum gelingt keine ernsthafte Konkurrenz zu Google.* eMail. July 2007. Online im Internet unter URL http://www.ub.uni-dortmund.de/listen/inetbib/msg34002.html. Abgerufen am 19.08.2007

[SBS98] Sander-Beuermann, Wolfgang ; Schomburg, Mario. *Internet Information Retrieval: The Further Development of Meta-Searchengine Technology.* 1998. Online im Internet unter URL http://www.metager.de/inet98/paper.html. Abgerufen am 30.06.2007

[Sch05] Scholze, Frank: Workflow für Online Hochschulschriften mit OPUS. In: *5. DissOnline Workshop*, 2005, Online im Internet unter URL http://www.dissonline.de/info/pdf/Frankfurt_Scholze.pdf. Abgerufen am 10.06.2007

[ST06] Speck, Hendrik ; Thiele, Frédéric P.: Suchmaschinen, Landschaften, Märkte und Transparenz. In: *3. Forum des SuMa-eV*, 2006, Online im Internet unter URL http://suma-ev.de/forum06/Presentation2006SumaLandschaftenMaerkteTransparenz.pdf. Abgerufen am 03.06.2007

[Sul07] Sullivan, Danny. *How Search Engines Rank Web Pages.* März 2007. Online im Internet unter URL http://searchenginewatch.com/showPage.html?page=2167961. Abgerufen am 19.08.2007

[Ter05] *Terrier Quickstart*, 2005. Online im Internet unter URL http://ir.dcs.gla.ac.uk/terrier/doc/quickstart.html. Abgerufen am 05.06.2007

[Tur07] Turner, Andrew: *OpenSearch Geo extension [draft]*, 2007. Online im Internet unter URL http://www.opensearch.org/Specifications/OpenSearch/Extensions/Geo/1.0/Draft_1. Abgerufen am 12.08.2007

[Web01] Weber, Gunnar: Integration von Datenbanken in Suchmaschinen bei unterschiedlichen Kooperationsgraden. In: *Informatik 2001: Wirtschaft und Wissenschaft in der Network Economy - Visionen und Wirklichkeit*

Bd. 1, Österreichische Computer Gesellschaft, 2001, S. 345–352. Online im Internet unter URL http://e-lib.informatik.uni-rostock.de/fulltext/2001/misc/GunnarWeber-Informatik2001-2001.pdf. Abgerufen am 04.06.2007

[Web06] Weber, Gunnar: *Techniken für Suchmaschinen zum Auffinden relevanter Informationseinheiten in Web-Datenbanken*, Universität Rostock, Diss., März 2006

[Web07] *Webhits Web-Barometer*. Juni 2007. Online im Internet unter URL http://www.webhits.de/deutsch/webstats.html#engines. Abgerufen am 03.06.2007

[Wik07a] *Application Programming Interface*. 2007. Online im Internet unter URL http://de.wikipedia.org/wiki/Application_Programming_Interface. Abgerufen am 20.08.2007

[Wik07b] Diverse (Hrsg.). *Atom (Format)*. 2007. Online im Internet unter URL http://de.wikipedia.org/wiki/Atom_%28Format%29. Abgerufen am 29.06.2007

[Wik07c] *Internetforum*. 2007. Online im Internet unter URL http://de.wikipedia.org/wiki/Internetforum. Abgerufen am 19.08.2007

[Wik07d] *Online-Community*. 2007. Online im Internet unter URL http://de.wikipedia.org/wiki/Online-Community. Abgerufen am 19.08.2007

[Wik07e] *PostScript*. 2007. Online im Internet unter URL http://de.wikipedia.org/wiki/PostScript. Abgerufen am 20.08.2007

[Wik07f] *Weblog*. 2007. Online im Internet unter URL http://de.wikipedia.org/wiki/Weblog. Abgerufen am 19.08.2007

[Wik07g] *Wiki*. 2007. Online im Internet unter URL http://de.wikipedia.org/wiki/Wiki. Abgerufen am 19.08.2007

[Zen06] *Zend_Search_Lucene*, 2006. Online im Internet unter URL http://framework.zend.com/manual/en/zend.search.lucene.html. Abgerufen am 09.06.2007

[Zen07] *Zend Framework API Documentation*. 2007. Online im Internet unter URL http://framework.zend.com/apidoc/core/. Abgerufen am 22.08.2007

Anhang A

Diagramme der Gesamtsysteme

In Abb. A.1 ist das Indizierungssystem als UML-Diagramm dargestellt, in Abb. das gesamte Suchsystem.
Neben den am Namen erkennbaren Zend_Search_Lucene-Klassen wurden zwei weitere nicht selbst entworfene Klassen genutzt: Die Klasse OpusFile und die Klasse Exception. *OpusFile* wurde von Stefan Freudenberg, Mitarbeiter des BSZ (Bibliotheksservice-Zentrum Baden-Würtemberg), entwickelt und gehört zur Umarbeitung von OPUS auf ein objektorientiertes Programmiermodell. Diese Umstellung ist zum Zeitpunkt der Entstehung dieser Arbeit noch sehr jung und noch nicht vollständig in OPUS integriert, so dass zugunsten der schnelleren Implementierung in der vorliegenden Arbeit teilweise noch alte Konzepte verfolgt wurden, die auf TUBdok noch nicht umgestellt sind.
Die Excetion-Klasse gehört zu PHP5 und stellt ein Gerüst zum Entwerfen eigener Exceptions bereit. Eine Exception kann in PHP aber auch auf diesem Gerüst beliebig verwendet werden, ohne dass extra eine Subklasse implementiert werden muss, solange diese nur die Grundeigenschaften und -methoden von Exception übernehmen soll.
Die Zend-Klassen sind teilweise nur auszugsweise abgebildet, um Platz zu sparen. In diesem Fall sind nur die im Wesentlichen benötigten Methoden und Attributwerte eingetragen.
Bei PHP5 können Konstruktoren auf zwei verschiedene Arten gebildet werden. Die erste Möglichkeit ist, wie auch schon in PHP4, die Methode nach der Klasse zu benennen. Die zweite Möglichkeit ist eine sogenannte *Magic Function* namens __construct. Beide Möglichkeiten sind äquivalent in ihren Auswirkungen, bei PHP5 ist jedoch die Konstruktorbildung über __construct übliche Regel, da dieser Konstruktor universeller einsetzbar ist und eine einheitliche Referenzierung sämtlicher Klassen ermöglicht. In den eigenen Klassen wurde in der Regel der __construct-Konstruktor benutzt.

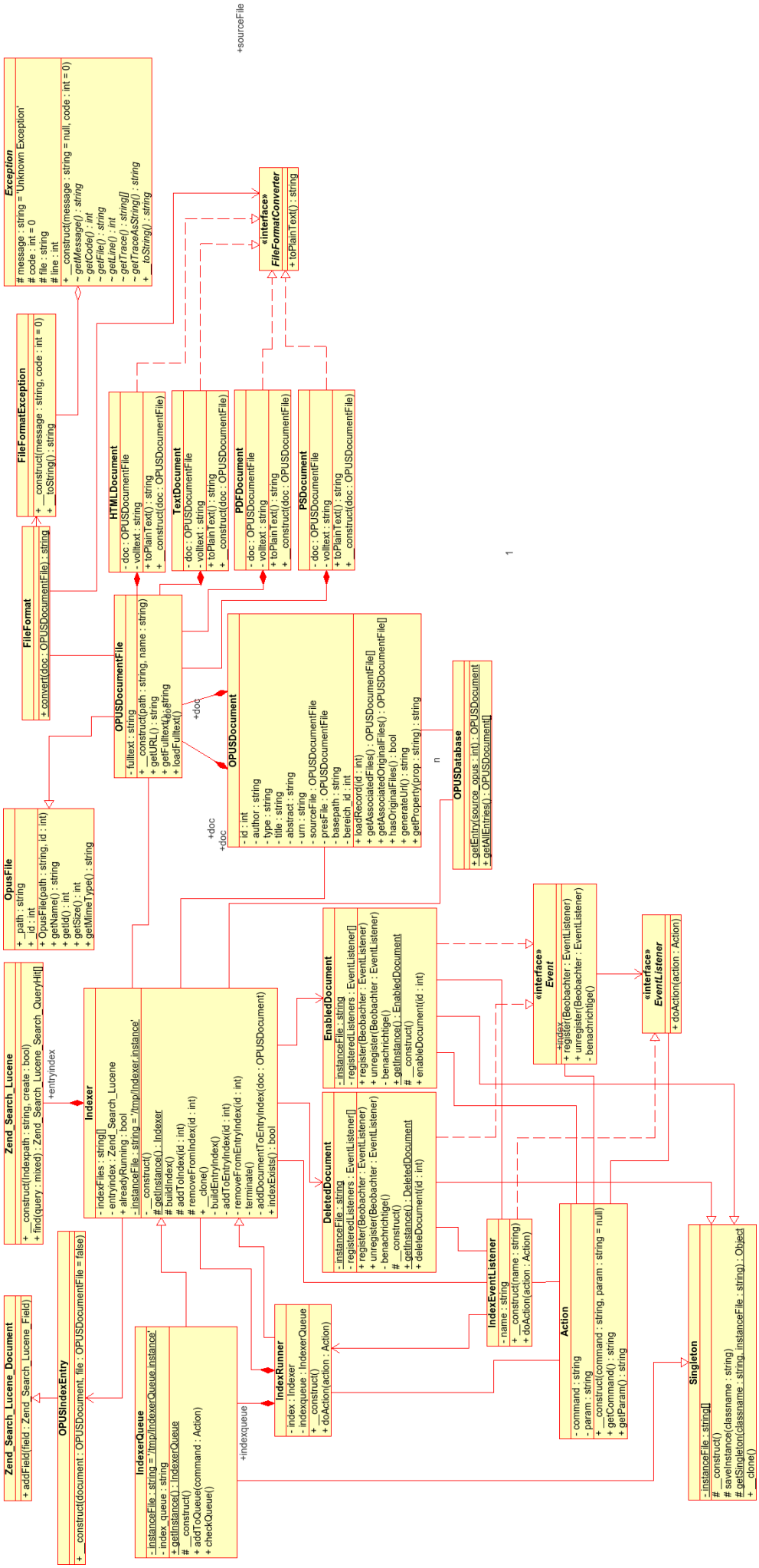

Abbildung A.1: UML-Klassendiagramm des Indexsystems

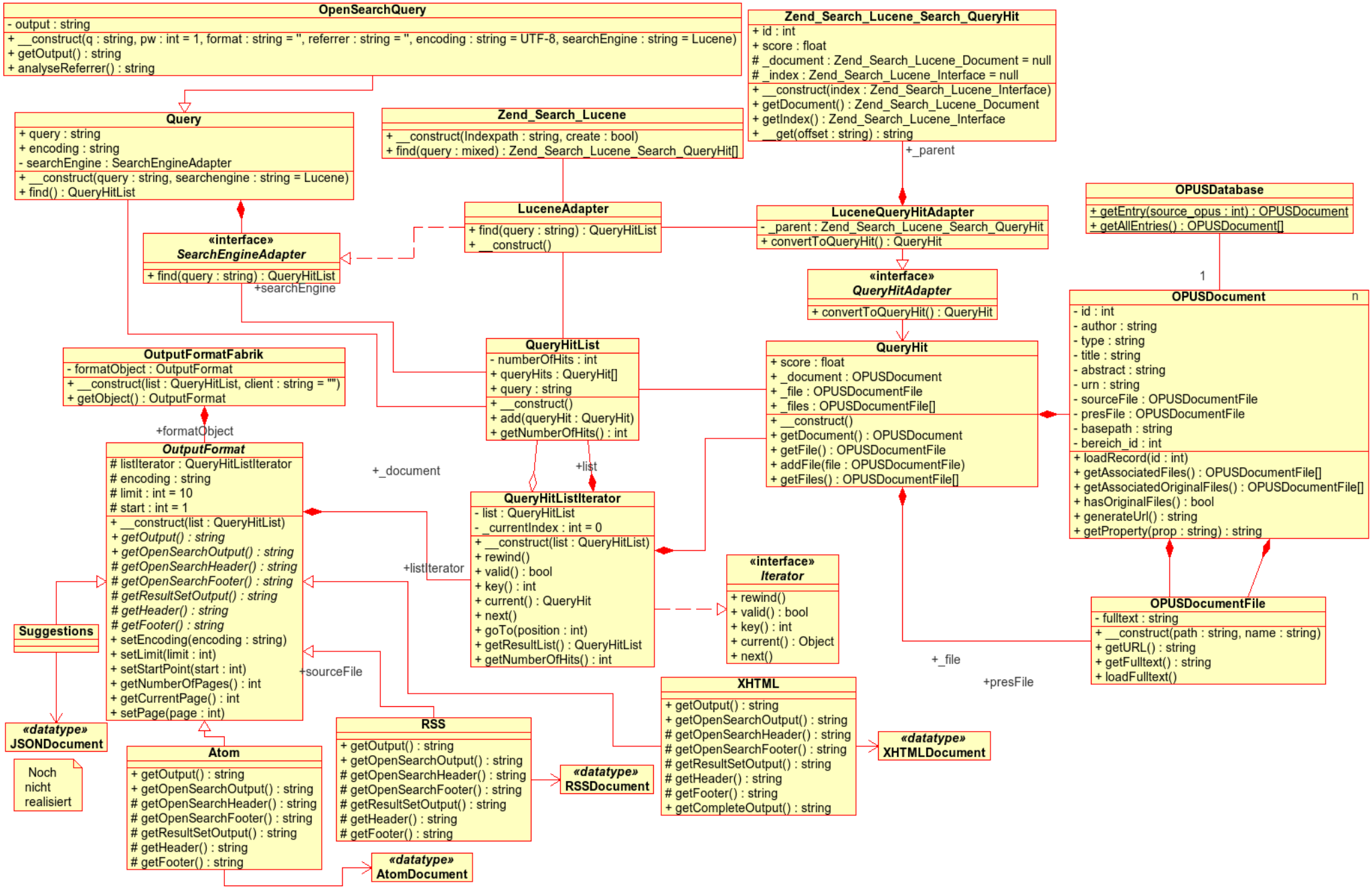

Abbildung A.2: UML-Klassendiagramm des Suchsystems

Anhang B

OpenSearch Description Document

Im Rahmen der Arbeit wurde ein OpenSearch Description Document für die Information-Retrieval-Schnittstelle erstellt. Dieses Description Document enthält die Spezifika des Servers, auf dem der Suchdienst läuft, die Beschreibung muss also, wenn andere OPUS-Instanzen auch eine OpenSearch-Suche anbieten möchten, jeweils angepasst werden. Das Description-Dokument des Dokumentenservers TUBdok ist hier abgedruckt. Das Original ist auf dem TUBdok-Testsystem öffentlich zugänglich unter `http://linws.b.tu-harburg.de:81/lucene/tubdok_opensearch.xml`

Listing B.1: OpenSearch Description File für TUBdok

```
1 <?xml version="1.0" encoding="UTF-8"?>
2  <OpenSearchDescription xmlns="http://a9.com/-/spec/
     opensearch/1.1/" xmlns:referrer="http://a9.com/-/
     opensearch/extensions/referrer/1.0/">
3    <ShortName>TUBdok Suchmaschine</ShortName>
4    <Description>Suchmaschine für den Dokumentenserver der
       Universitätsbibliothek der TU Hamburg Harburg</
       Description>
5    <Tags>Dokumentenserver Universität</Tags>
6    <Contact>o.marahrens@tu-harburg</Contact>
7    <Url type="application/atom+xml" template="http://linws.b
       .tu-harburg.de:81/lucene/opensearch.php?format=atom&
       amp;q={searchTerms}&amp;pw={startPage?}&amp;nr={
       itemsPerPage?}&amp;src={referrer:source?}"/>
8    <Url type="application/rss+xml" template="http://linws.b.
       tu-harburg.de:81/lucene/opensearch.php?format=rss&amp;
       q={searchTerms}&amp;pw={startPage?}&amp;nr={
       itemsPerPage?}&amp;src={referrer:source?}"/>
```

```
9    <Url type="text/xhtml" template="http://linws.b.tu-
         harburg.de:81/lucene/opensearch.php?q={searchTerms}&
         amp;pw={startPage?}&amp;nr={itemsPerPage?}&amp;src={
         referrer:source?}"/>
10   <Url type="text/html" template="http://linws.b.tu-harburg
         .de:81/lucene/search.php?q={searchTerms}&amp;nr={
         itemsPerPage?}&amp;pw={startPage?}"/>
11   <!-- Noch nicht unterstuetzt <Url type="application/x-
         suggestions+json" template="http://linws.b.tu-harburg.
         de:81/lucene/opensearch.php?format=suggestions&amp;q={
         searchTerms}"/> -->
12   <LongName>Suchmaschine für den Dokumentenserver TUBdok</
         LongName>
13   <Image height="64" width="64" type="image/png">http://
         linws.b.tu-harburg.de:81/search/opensearch.png</Image>
14   <Image height="16" width="16" type="image/vnd.microsoft.
         icon">http://linws.b.tu-harburg.de:81/favicon.png</
         Image>
15   <Query role="example" searchTerms="xml" />
16   <Developer>Oliver Marahrens</Developer>
17   <Attribution>
18     Search data provided by University Library TUHH, All
         Rights Reserved
19   </Attribution>
20   <SyndicationRight>open</SyndicationRight>
21   <AdultContent>false</AdultContent>
22   <Language>de-de</Language>
23   <OutputEncoding>UTF-8</OutputEncoding>
24   <InputEncoding>UTF-8</InputEncoding>
25 </OpenSearchDescription>
```