Technische Universität Graz

Masterarbeit

Anwendungsbereiche von Suchmaschinen

Erweiterung der Such-Funktionalität im Austria-Forum

Dieter Steininger

27. August 2012

Betreuer
Assoc.Prof. Dipl.-Ing. Dr.techn. Denis Helic,
Institut für Wissensmanagement

Geschrieben in
LaTeX unter Verwendung von
TeXnicCenter 2.0 Alpha 3, Build 1118

Vorlage
LaTeX Template von Karl Voit, LaTeX@TUG
https://github.com/novoid/LaTeX-KOMA-template
http://latex.tugraz.at/

Struktureller Leitfaden
„Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science"
Keith Andrews [And11]

Inhaltsverzeichnis

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am 27.8.2012 ..... (Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

2012/08/27 ..... date

..... (signature)

Kurzfassung

In unserer technologisch hoch entwickelten Gesellschaft ist angesichts der Flut von digitalen Daten ein Leben ohne moderne Suchmaschinen kaum noch vorstellbar. Diese Arbeit widmet sich den theoretischen Aspekten solcher Applikationen und deren Umsetzung in einem konkreten Anwendungsfall.

Die ersten beiden Kapitel bieten einen kurzen Einblick in die geschichtliche Entwicklung von Suchmaschinen und schaffen einen umfassenden Überblick über diesen Bereich aus dem Gebiet der Informationsrückgewinnung. In Kapitel 3 wird die Open Source Bibliothek *Lucene* vorgestellt, mit deren Hilfe Suchapplikationen entwickelt werden können. Der anschließende Abschnitt bettet die theoretischen Aspekte in einen konkreten Anwendungsfall — das Austria-Forum — ein. In den Kapiteln 5 bis 7 werden die vorgenommenen Modifikationen der Such-Funktionalität in diesem Forum vorgestellt. Dazu gehören Adaptierungen der Indexierung für deutschsprachige Quelldokumente, Verbesserung der Bedienerfreundlichkeit und Erweiterungen bei der benutzerdefinierten Suche wie beispielsweise Ähnlichkeitssuche und die Verwendbarkeit von Abkürzungen. Weiters wird die Integration des deutschen Wortnetzes *GermaNet* zur Synonymsuche vorgestellt, um den Anwender bei einer Umformulierung seiner Anfrage zu unterstützten.

Das abschließende Kapitel 8 spiegelt die praktischen Erfahrungen wider, die während der Umsetzung zuvor genannter Erweiterungen gesammelt wurden. Beigefügte Ausblicke zeigen Potenziale auf, die bei der *„Suche im Austria-Forum“* eventuell noch ausgeschöpft werden könnten und bieten dadurch mögliche Anknüpfungspunkte für weitere Tätigkeiten in diesem Bereich.

Schlüsselwörter:
Suchmaschinen, Lucene, Austria-Forum, Indexierung und Suche bei deutschsprachiger Datenbasis, Synonymsuche, GermaNet, Ähnlichkeitssuche

Abstract

We live in an information era and have to handle a flood of digital data. It is hardly possible to imagine a life without modern search engines in such an environment. This thesis deals with the technically background of search applications and its appliance in a concrete application.

The first two chapters give a short insight into the historical development of search engines and offer a comprehensive overview of this field of Information Retrieval. Chapter 3 introduces *Lucene*, an Open Source Library for the development of search engines. The theory gets embedded in a practical implementation — the Austria-Forum — in the following sections. Chapters 5 to 7 describe the implemented enhancements of search functionality in this application. These are: improved indexing for german documents, improved user-friendlyness and enhancements in user-defined search e.g. fuzzy search and an implemented shortcut list. Furthermore the linking of the german word net *GermaNet* for searching for synonyms is described. This functionality helps the user rephrasing his query by offering synonymous words.

The final chapter 8 summarizes the practical experiences gained during the implementation of the mentioned enhancements. Additional outlooks show unused capabilities and could be a stimulus for future work on search functionalities in the Austria-Forum.

Keywords:
search engines, Lucene, Austria-Forum, Indexing and search for german documents, synonyms search, GermaNet, fuzzy search

Danksagung

Diese Masterarbeit ist am Institut für Wissensmanagement an der Technischen Universität Graz entstanden. Die Mitarbeit am Austria-Forum, im Speziellen an den darin enthaltenen Suchfunktionalitäten, war die Basis für den praktischen Teil. An dieser Stelle möchte ich mich bei Herrn Em.Univ.-Prof. Dr.phil. Hermann Maurer für die Möglichkeit zur Mitentwicklung am Austria-Forum bedanken.

Ein besonderes Dankeschön ergeht an meinen Betreuer, Herrn Assoc.Prof. Dipl.-Ing. Dr.techn. Denis Helic. Seine Unterstützung hat es mir erlaubt, ungeachtet meines Umzugs nach Wien, mein Studium an der TU Graz zu beenden. Ohne sein Entgegenkommen und seiner Bereitschaft zur unbürokratischen und multimedialen Zusammenarbeit wäre das nicht möglich gewesen.
Vielen Dank dafür!

Der größte Dank gilt meiner Familie, die trotz der langen Dauer meines weitestgehend berufsbegleitenden Studiums (fast) nie den Glauben an mich verloren hat. Ohne die Hilfe meiner Eltern Franz und Elisabeth wäre ein Hochschulstudium ungleich schwieriger zu bewältigen gewesen. Sowohl die aufmunternden als auch die ermahnenden Worte meiner Eltern und meiner Schwester Karin haben mich immer wieder dazu bewegt, das Ziel nicht aus den Augen zu verlieren. Wenn ich doch einmal längerfristig das Studium beiseite liegen ließ, haben sie mir den Sinn wieder in Erinnerung gerufen und mich zum Weitermachen motiviert. Dafür haben sie sicherlich viel Kraft und Geduld benötigt.
Ich danke ihnen von ganzem Herzen!

Meinen Freundeskreis möchte ich ebenfalls nicht unerwähnt lassen. Viele langjährige Weggefährten haben mir oftmals Mut zugesprochen und mir durch manche schwere Stunde geholfen. Etliche Wege und Erledigungen wurden mir abgenommen, damit ich diversen Pflichten fristgerecht nachkommen konnte ohne dafür immer stundenlange Autofahrten und verbrauchte Urlaubstage in Kauf nehmen zu müssen. Stets haben sie Verständnis gezeigt, wenn ich private Vergnügungen gegenüber dem Studium und der Arbeit hinten angestellt habe.
Dieser Rückhalt wird mir immer in wohlwollender Erinnerung bleiben!

1 Einführung

1.1 Wozu Suchmaschinen?

Die Sumerer gelten als älteste bekannte Hochkultur und Erfinder der Schrift im 4. Jahrtausend vor Christus. Die auf Tontafeln erhaltenen Keilschriften hatten vorwiegend wirtschaftliche und administrative Inhalte wie etwa Aufzeichnungen über Zahlungsvorgänge oder Besitztümer. Sie wurden oft in Palastarchiven gelagert und schon damals gab es Ansätze zur Erschließung dieser Archive. Tafelkataloge und die Beschriftung von Sammelbehältnissen mit den Titeln oder Stichworten der darin gelagerten Tontafeln sind Beispiele dafür.

Der Wunsch Informationen zu sammeln, archivieren und möglichst einfach wieder aufzufinden war wohl Triebfeder für diese Entwicklungen. Und auch heute, Jahrtausende später, unternimmt die Menschheit vielerlei Anstrengungen um dem selben Ziel zu folgen. Das Verlangen Daten zu archivieren und organisieren ist also ein stetiger Begleiter der Menschheitsgeschichte.

Zur Zeit des zweiten Weltkriegs war ein Großteil der Wissenschaftler mit Themen wie Kriegsführung und Kampfmitteln beschäftigt. Der Amerikaner Vannevar Bush koordinierte die Arbeit von rund 6000 Personen in diesem Bereich. Nach Ende des Krieges machte er sich Gedanken darüber, in welchen Bereichen diese Wissenschaftler künftig arbeiten würden, welche Entwicklungen absehbar waren und wie die Arbeitsweise und Interaktion künftiger Wissenschaftler aussehen könnten. Der effizienten Speicherung und Wiederauffindbarkeit von Informationen schenkte Bush dabei besondere Aufmerksamkeit.

Memex

Im Juli 1945 erschien sein Artikel *„As We May Think“* im Atlantic Magazine. [Bus45] Zu dieser Zeit waren Technologien wie Faksimile und Mikrofilm noch unausgereift und Ideen zur Spracherkennung und -synthetisierung steckten in den Kinderschuhen. Bush malte in seinem Artikel einige Visionen aus, die ihn zu einer Maschine führten, die er *Memex* taufte. Zweck dieser — im Wesentlichen einem Schreibtisch ähnelnden — Maschine sollte die effiziente Speicherung von Informationen, die Erstellung von Assoziationen zwischen ihnen und die Übertragbarkeit von Wissen sein. Gegenüber den bis dahin verwendeten

hierarchischen Indexe und Ablagesystemen stellten vor allem die direkten Verknüpfungen zwischen Informationen — die Links — eine wesentliche Annäherung an die menschliche Denkweise dar. Das Vernetzen von Informationen, die Nachvollziehbarkeit von Gedankengängen und dadurch das rasche Auf- und Wiederfinden von Daten waren die zentralen Ansinnen, die durch Kombination verschiedener Technologien ermöglicht werden sollten.

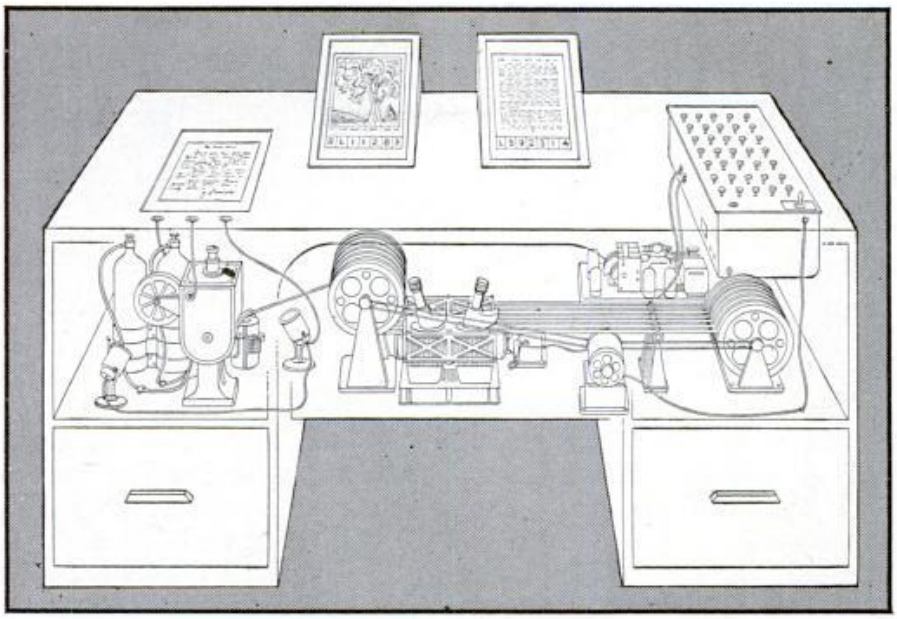

Abbildung 1.1: *Memex - Illustration der fiktiven Maschine nach den Ideen von Vannevar Bush*
Quelle: Life Magazine, Ausgabe vom 10. September 1945, Seite 123

Das Konzept des Memex wird heutzutage oft als Inspiration für Dinge wie Personal Computer oder Hypertext genannt. Die meisten Konzepte dieser fiktiven Maschine sind heute allgegenwärtig verwirklicht und die Grundgedanken — Verknüpfungen zwischen Informationen zu schaffen und diese möglichst rasch wieder zu finden — sind nach wie vor angemessen und aktuell.

Index

Doch wie sollen Informationen gefunden werden? Der einfachste Ansatz wäre es, alle Daten durchzusehen, bis man das Gesuchte erspäht. Es ist offensichtlich, dass dieser Ansatz mit steigender Größe der archivierten Informationen — der Datenbasis — zunehmend unpraktikabel wird. Abhilfe bietet die Erstellung eines *Index* in einer Form, die das langsame, sequenzielle Durchstöbern der gesamten Datenbasis eliminiert und eine schnellere Suche ermöglicht. Wozu hätten die Sumerer sonst Tafelkataloge angelegt?

Die Informationssuche basiert daher auch im Zeitalter moderner Suchmaschinen auf einem Index, der sich in seiner Form jedoch von den Baumstrukturen der vergangenen Jahrtausende abhebt. Die Entwicklung von einer hierarchi-

schen zu einer assoziativen Verknüpfung von Daten hat die Notwendigkeit zur Folge, Informationen auf alternative Weisen zu erschließen.

> *„Suchmaschinen sind zu unverzichtbaren Navigationsinstrumenten für die dynamische und expandierende Informationslandschaft des Internet geworden. Da das Internet weder über einen eigenen Index noch ein Katalogisierungssystem verfügt, erfüllen Suchmaschinen eine zentrale Aufgabe: digitale Information zugänglich und damit nutzbar zu machen.“* Konrad Becker und Felix Stalder [BS09, Seite 7]

Im Informationszeitalter steht zusehends das Auffinden von digitalen Informationen aus unterschiedlichsten Quellen gegenüber dem eigenständigen Verknüpfen selbst zusammengetragener Daten im Vordergrund. Mit der Explosion der verfügbaren Datenmenge und dem Aufkommen des Internet begann daher auch die Ära der Suchmaschinen.

1.2 Geschichtliche Entwicklung

Das Internet als Nachfolger des Arpanet[1] verband ursprünglich Universitäten und Forschungseinrichtungen miteinander, ehe es 1990 öffentlich zugänglich gemacht wurde. Die Inhalte waren praktisch ausschließlich von computerwissenschaftlicher und technischer Natur und lagen häufig in Form von FTP[2] Archiven vor.

Archie

1987 wurde ein Team der McGill Universität Montreal damit beauftragt, Möglichkeiten zur Anbindung der Universität an das Internet zu untersuchen. Weder Geld noch politische Unterstützung waren für dieses experimentelle Unterfangen vorhanden. Ansporn lieferte lediglich die Überzeugung, dass es von Nutzen sein könnte Zugang zu dieser wissenschaftlichen Gemeinschaft zu haben. Im Jahr 1989 wurde Alan Emtage damit beauftragt die Adressen und Inhalte der verfügbaren FTP-Archive zusammenzutragen.

Emtage schrieb dafür ein Skript um die Inhaltsverzeichnisse der FTP Verzeichnisse automatisiert zu beziehen und in einer Datei lokal abzuspeichern. In dieser Datei konnte dann mittels Standardbefehlen des Unix Betriebssystems nach

1 Advanced Research Projects Agency Network, ab 1962
2 File Transfer Protocol

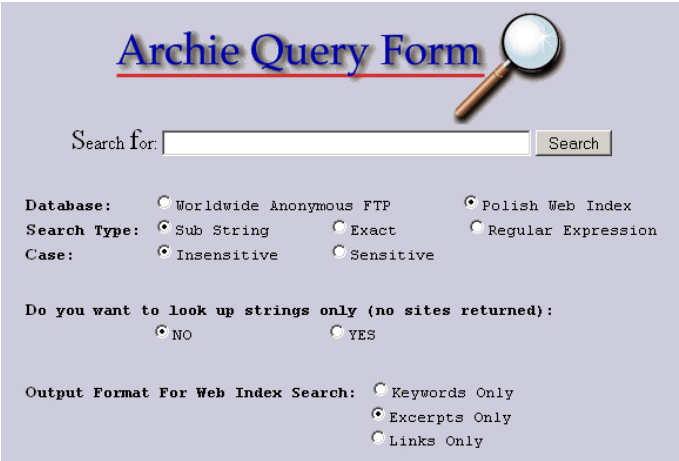

Abbildung 1.2: *Archie - moderne Benutzeroberfläche einer noch aktiven Archie-Implementation mit Suchoptionen*
Quelle: http://archie.icm.edu.pl/archie_eng.html

Namen von FTP-Dateien und deren Adressen im Netzwerk gesucht werden. Als diese Möglichkeit publik wurde häuften sich die Anfragen von Personen, die dadurch hofften gewünschte Informationen zu finden. Um diese Anfragen nicht manuell bearbeiten zu müssen, wurde 1990 ein telnet-Zugang geschaffen um den Nutzern direkten Zugang zu gewähren und *Archie*[1] war geboren. [Deuoo]

Archie gilt somit als erste Suchmaschine und Vorreiter der Internet-basierenden Informationsdienste. Weitere Suchmaschinen entwickelten sich in Windeseile und Archie wurde mit der Zeit bedeutungslos.

Weitere Entwicklung

Der folgende Abriss einiger populärer, früher Vertreter soll einen Eindruck über die Entwicklung von Suchmaschinen vermitteln, die heutzutage längst nicht mehr „nur“ im Internet operieren. [Helo8], [Wal], [Sun]

Menügesteuerte Oberfläche

1991 wurde an der Universität von Minessota der Informationsdienst *Gopher* entwickelt. An der Universität von Nevada wurde 1992 *Veronica*[2] entwickelt, die diesen Dienst nutzte um große Datenmengen über eine menügesteuerte Oberfläche abzufragen. Dabei handelte es sich vorwiegend um Textdateien. Bei der Abfrage konnten bereits die booleschen Operatoren *Und*, *Oder* und *Nicht* verwendet werden.

1 „Archie“ steht für „Archive“
2 Very Easy Rodent-Oriented Netwide Index to Computerized Archives

Erste Robots

Kurz nach dem Aufkeimen des neu entstandenen *World Wide Web* stellte Mathew Gray im Juni 1993 mit dem *World Wide Web Wanderer* den ersten Robot[1] vor. Ursprünglich gedacht um aktive Webseiten zu zählen und das Wachstum des *WWW* zu messen, wurde dieses Programm bald umfunktioniert um Adressen — die URL's[2] — zu sammeln. Wie bei den meisten frühen Robots war die Implementierung nicht vollends ausgeklügelt und diese Pioniere verursachten somit bald erhebliche Probleme aufgrund ihres Ressourcenverbrauchs.

Ergebnisreihung

Im selben Jahr gingen *Aliweb*[3], *Jumpstation*, *World Wide Web Worm* und *RBSE Spider*[4] online. Aliweb erlaubte den Benutzern eine Beschreibung ihrer eigenen Seiten hochzuladen und kam auf diese Art ohne Robot aus. Jumpstation und World Wide Web Worm lieferten die Ergebnisse in der Reihenfolge ihres Auffindens aus, wohingegen RBSE Spider als erste Suchmaschine eine Sortierung der Ergebnisse durchführte.

Volltext-Indexierung

Im April 1994 wurde der von Brian Pinkerton an der Universität vom Washington entwickelte *WebCrawler* in Betrieb genommen. Dabei handelte es sich um die erste Suchmaschine, die nicht nur Adresse, Titel und einige Metadaten indexierte, sondern den gesamten Inhalt — also den Volltext — der besuchten Seiten durchsuchbar machte.

Nähe der Begriffe

Im gleichen Jahr startete *Lycos* an der Carnegie Mellon University. Neben der Worthäufigkeit der Treffer wurde erstmals auch die Nähe der Suchbegriffe im Text in die Reihung mit einbezogen. 1996 war Lycos mit 60 Millionen indexierten Dokumenten die größte Suchmaschine seiner Zeit.

kommerzielle Anwendung

Die bisher genannten Suchmaschinen entstammten allesamt verschiedenen Universitäten. 1995 gingen mit *Infoseek*, *Alta Vista* und *Architext*[5] die ersten Suchmaschinen ans Netz, die von kommerziellen Unternehmen betrieben wurden.

Web Verzeichnis

David Filo und Jerry Yang führten 1994 eine Liste ihrer bevorzugten Seiten im Netz. Der Umfang der Liste erforderte bald eine Reorganisation zu einem durchsuchbaren Verzeichnis, bekannt unter dem Namen *„Yahoo!“*. Neben den Adressen stellten sie dabei eine selbst erstellte Beschreibung der Inhalte zur Verfügung. Yahoo! wuchs und Betreiber kommerzieller Internet-Seiten bezahlten bald für die Aufnahme in das Verzeichnis. Angeblich aus Frust über die lan-

1 siehe Abschnitt 2.4.1 auf Seite 19
2 Uniform Resource Locator
3 Archie-Like Indexing of the Web
4 Repository-Based Software Engineering Spider
5 später *Excite*

gen Wartezeiten bei der Aufnahme in Yahoo! gründete ein Kreis befreundeter Webmaster das *Open Directory Project ODP*[1], das bis heute das größte Internet-Verzeichnis ist.

Im Gegensatz zu Suchmaschinen, deren Index größtenteils oder ausschließlich automatisiert aufgebaut wird, werden die Web Verzeichnisse durch Menschen gewartet. Einerseits passiert dabei eine mehr oder weniger strenge redaktionelle Prüfung, andererseits entstehen enorme Kosten. Aufgrund des Personaleinsatzes können solche Verzeichnisse in Umfang und Wachstum mit Suchmaschinen nicht schritt halten.

Google und bing

1996 beginnen Larry Page und Sergey Brin mit der Entwicklung der Suchmaschine *BackRub*. Wesentliche Merkmale sind dabei der Ansatz zur Ergebnissortierung und die Hardware in Form vieler Standard-Computer. 1998 geben sie ihrer Suchmaschine einen neuen Namen: *Google*. Dieser Suchmaschine wird aufgrund ihrer Popularität ein eigener Abschnitt (2.7.2) in dieser Arbeit eingeräumt. Microsoft® startet 1998 seine Suchmaschine *MSN Search*, und bringt 2006 und 2009 deren Nachfolger *LiveSearch* und *bing* auf den Markt.

Diese Liste stellt nur einen kleinen Auszug aus der enormen Entwicklung der letzten 25 Jahre dar und etliche — zumindest zwischenzeitlich — populäre und erfolgreiche Vertreter bleiben ungenannt. Dennoch lässt sich erahnen wie rapid die Entwicklung in diesem Feld voran schreitet.

1.3 Motivation

Suchmaschinen sind mittlerweile in viele Internet-Auftritte und elektronische Foren von kommerziellen und privaten Betreibern integriert. Auch im Austria-Forum[2] ist eine Suche realisiert. Bei der ursprünglichen Implementation zeigte die praktische Verwendung jedoch einige Verbesserungsmöglichkeiten auf. Diese Punkte lieferten den Anreiz für Erweiterungen und Änderungen an der Suche im Austria-Forum und bilden die Basis des praktischen Teils dieser Arbeit.

Eingabe der Anfrage: Auf der Ergebnisseite wurde dem Benutzer die Möglichkeit geboten, seine ursprüngliche Anfrage zu verändern oder genauer zu spezifizieren. Dabei wurde vom Benutzer jedoch die Eingabe in einer syntaktischen Form erwartet, die unmittelbar von Lucene[3] verarbeitet werden konnte. Die Darstellung und Eingabe der Anfrage in dieser speziellen

1 auch bekannt als *directory mozilla dmoz*
2 siehe Kapitel 4
3 siehe Kapitel 3 und Abschnitt 5.2.1

Form wirkte auf Laien eher abschreckend als einladend. Ein Ziel der praktischen Verbesserungen war es, dem Anwender eine intuitive Interaktion mit der Suche zu erlauben und die technische Realisierung besser im Hintergrund zu verstecken.

Darstellung der Ergebnisse: Neben den Ergebnissen wurden Balken dargestellt, die die Relevanz der jeweiligen Treffer optisch veranschaulichen sollten. Die Balkenlängen waren dabei stets von der Gesamtheit der aktuell dargestellten Resultate abhängig und damit nicht über mehrere Ergebnisseiten oder gar verschiedene Suchanfragen hinweg vergleichbar. Eine Vereinheitlichung dieser Anzeige war ebenfalls erwünscht.

Sprache der Quelldokumente: Ein wesentliches Manko stellte der Umstand dar, dass die Suche signifikante Schwachstellen bei Besonderheiten der deutschen Sprache aufwies. Vor allem für Begriffe mit Umlauten oder *ß* wurden nur unbefriedigende Ergebnisse erzielt. Da die Beiträge im Austria-Forum größtenteils deutschsprachig sind, sollte auch die Suche besser auf eine deutsche Datenbasis abgestimmt werden.

Ähnlichkeitssuche: Vor den Modifikationen war die Suche intolerant gegenüber Tippfehlern. Wenn Suchbegriffe nicht korrekt eingegeben wurden weil etwa Buchstaben irrtümlich vertauscht wurden, konnten zumeist keine Treffer erzielt werden. Der Wunsch nach einem fehlertoleranten Verhalten war Motivation für dieses Novum.

Abkürzungen: Viele Abkürzungen bedürfen für Menschen im täglichen Gebrauch und jeweiligen Kontext keinerlei weiteren Erklärungen. Ein „St.“ in Ortsnamen wie „St.Ruprecht“ wird beispielsweise unwillkürlich als *Sankt* gelesen. Aus dieser Selbstverständlichkeit ergibt sich der Bedarf, dass auch bei der Suche derart gebräuchliche Abkürzungen im System hinterlegt und vom Anwender genutzt werden können. Bislang war dies im Austria-Forum nicht möglich.

Umformulierung der Anfrage: Suchen nach unterschiedlichen Begriffen liefern im Allgemeinen auch unterschiedliche Resultate — auch wenn die einzelnen Begriffe inhaltlich die selbe Bedeutung haben. Für Benutzer ist jedoch oft der thematische Inhalt wichtiger als die Erscheinung eines speziellen Wortes. Daraus ergibt sich die Idee, dem Anwender bedeutungsgleiche Begriffe vorzuschlagen und ihn damit bei einer Umformulierung seiner Anfrage zu unterstützen. Das war der Anstoß um ein deutschsprachiges Wortnetz anzubinden und damit eine Synonymsuche zu realisieren. Die Darbietung von bedeutungsgleichen Begriffen stellt ebenfalls eine wesentliche funktionale Erweiterung dar.

Die Summe dieser Kritikpunkte und Bedürfnisse bildete die Motivation zur Modifikation der Suche im Austria-Forum. Die praktisch orientierten Kapitel dieser Arbeit erläutern wie diese Anforderungen umgesetzt wurden. Die Auswirkungen der vorgenommenen Änderungen und Erweiterungen werden ebenfalls in den späteren Kapiteln ausführlich veranschaulicht.

Aufbauend auf der begründeten Sinnhaftigkeit und der geschichtlichen Entfaltung moderner Suchmaschinen sollen zuerst die theoretischen Grundlagen zu dieser Thematik geklärt werden. Sie sind in späterer Folge für das Verständnis der praktischen Anwendungen und Adaptierungen der Suche im Austria-Forum notwendig.

2 Moderne Suchmaschinen

Im vorliegenden Kapitel werden zuerst einige Klassifizierungen von Suchmaschinen vorgestellt. In weiterer Folge werden die wesentlichen Anforderungen diskutiert und eine allgemeingültige Architektur zur Bewältigung dieser Herausforderungen erläutert. Ausgehend davon werden die beiden elementaren Prozesse — Indexierung und Suche — eingehender betrachtet. Ein Abschnitt zur Evaluierung und einige Beispiele moderner Suchmaschinen runden dieses Kapitel ab.

Obwohl versucht wird einen allgemeingültigen Überblick zu vermitteln, ist eine gewisse Affinität zu Internet-Suchmaschinen in Teilen dieses Kapitels nicht zu leugnen. Suchapplikationen im *World Wide Web* sind sehr populär und haben in mancherlei Hinsicht maximale Anforderungen zu bewältigen. So müssen sie mit der größten denkbaren Datenbasis zurande kommen und gleichzeitig für ganze Heerschaften von simultanen Benutzern möglichst gute Performanz erzielen. Der Konkurrenzdruck auf die Qualität ist dabei enorm. Die verwendeten Konzepte sind daher mehr oder weniger stark abgewandelt auch auf die anderen Klassen der Suchmaschinen anwendbar.

2.1 Klassifizierung

Suchmaschinen lassen sich unter den unterschiedlichsten Gesichtspunkten einteilen. Art, Größe und Dateitypen der Datenbasis, der monetäre Standpunkt oder die technische Umsetzung in Hinblick auf Programmiersprache oder Architektur sind nur einige Beispiele. Nachstehend werden jene Kategorisierungen kurz erläutert, die im Rahmen dieser Arbeit von besonderem Interesse sind.

Zunächst ist eine übergeordnete Klassifizierung von *„Suchanwendungen im erweiterten Sinn“* nach ihrer Wesensart in die folgenden Typen möglich.

Verzeichnisse

Web Verzeichnisse ermöglichen die Suche in von Menschenhand erstellten und gewarteten Indexen beziehungsweise Katalogen. Die Quellen[1] werden dabei entweder durch ein Redaktionsteam klassifiziert und in das Verzeichnis eingetragen, oder durch Betreiber beziehungsweise Benutzer dezidiert angemeldet. So entsteht eine hierarchische Struktur, in der man über thematische Kategorien und Unterkategorien zu den Quellen navigieren kann. Manuell gewartete Systeme haben generell Schwierigkeiten dem rasanten Wachstum des Internet nachzukommen.

Index-basierende Suchmaschinen

Sie erstellen ihren Index automatisiert und zumeist ohne jegliches menschliche Zutun. Somit können sie dem raschen Anstieg der Mengen von digitalen Daten Rechnung tragen und skalieren sehr gut. Das vorliegende Kapitel setzt den Fokus auf diese Anwendungen.

Meta-Suchmaschinen

Solche Applikationen nehmen ihrerseits keine eigenständige Indexierung vor. Sie verteilen die Anfrage über Applikationsschnittstellen an mehrere Suchmaschinen, fassen deren Ergebnisse zusammen und präsentieren dem Anwender eine konsolidierte Liste der Treffer.

Als Datenbasis steht damit die Summe der Indexe der konsultierten Suchmaschinen zur Verfügung. Das ist besonders bei sehr selten vorkommenden Suchbegriffen von Vorteil. Weiters muss kein eigener Index erstellt werden, was die Implementation erheblich vereinfacht. Durch die Verteilung der Anfrage auf mehrere Suchmaschinen, das Warten auf deren Resultate und anschließende Konsolidieren der Einzelergebnisse, ergibt sich jedoch eine längere Antwortzeit.[2]

Die Suche im Austria-Forum ist durch eine index-basierende Suchmaschine realisiert. Diese Arbeit legt daher das Hauptaugenmerk auf solche Anwendungen, die im Weiteren kurz hin als „Suchmaschinen“ tituliert werden. Sie können nach den folgenden Gesichtspunkten weiter unterteilt werden:

1 Als Quellen werden hier eher gesamte Websites verstanden als einzelne Dokumente.
2 siehe Seite 14

2.1.1 Klassifizierung nach Offenlegung des Quellcodes

Proprietär oder quellfrei?

Auch bei den Suchmaschinen gibt es — wie in nahezu allen IT-Bereichen — zwei konträre Strategien: Die *proprietären Suchmaschinen*[1] stehen den *Open Source Implementationen*[2] gegenüber.

Obwohl aus diesen kontroversen Ansätzen geringfügige Anforderungsunterschiede bezüglich der Architektur resultieren, sind die Kernaufgaben weitestgehend identisch. Der Hauptgrund um bei Suchmaschinen den Quellcode unter Verschluss zu halten liegt im wirtschaftlichen Interesse der Betreiber. Umgangssprachlich werden Closed-Source Implementationen daher oft auch als *kommerzielle Suchmaschinen* bezeichnet. Das bedeutet jedoch weder dass proprietäre Suchmaschinen für den Benutzer zwangsläufig kostenpflichtig sind, noch dass Open Source Anwendungen nicht kommerziell eingesetzt werden dürfen. Im Rahmen dieser Masterarbeit liegt der Fokus jedoch auf den quellfreien Implementationen. Das begründet sich durch zwei Hauptaspekte:

Verfügbarkeit von Informationen ist bei proprietären Anwendungen naturgemäß nur bedingt gegeben. Unternehmen, deren Wertschöpfung und Gewinn zu einem wesentlichen Teil darauf beruhen *wie* sie eine Problemstellung technisch lösen, hüten diese Implementationsdetails dementsprechend gut. Hier ist man also gezwungen auf die Richtigkeit und Vollständigkeit der öffentlich zugänglichen Informationen zu vertrauen. Bei Open Source Projekten hat man hingegen die Möglichkeit die Umsetzung bis ins letzte Detail selbständig nachzuvollziehen und zu verifizieren.

Die technische Umsetzung des Austria-Forums ist eine gegebene Randbedingung für den praktischen Teil dieser Arbeit. Die Verwendung von Open Source Komponenten und der Einsatz von Lucene rechtfertigen ebenfalls das verstärkte Augenmerk auf quellfreie Suchmaschinen.

Proprietäre Anwendungen seien hier dennoch nicht gänzlich außer Acht gelassen. Einerseits dienen Verweise darauf zur Abrundung dieser Arbeit, andererseits sind — wie eingangs erwähnt — viele Anforderungen und Konzepte gleichermaßen für beide Strategien zutreffend.

1 auch *Closed Source* oder umgangssprachlich *kommerzielle* Suchmaschinen genannt

2 Unter Open Source versteht man quellfreie Software, die unter einer Palette von Lizenzen steht, die den Quellcode öffentlich zugänglich machen und Weiterentwicklung fördern.

2.1.2 Klassifizierung nach Datenbasis

Eine weitere Unterteilung der modernen Suchmaschinen basiert auf dem zu durchsuchenden Datenbestand:

Desktop oder *World Wide Web*?

Internet-Suchmaschinen — mit dem *World Wide Web* als Datenbasis — kommen vielen Computer-Benutzern wohl zuerst in den Sinn. Schließlich würde man in den Weiten dieses globalen Mediums praktisch nur mehr die wenigsten Informationen ohne maschinelle Hilfe finden.

Unternehmensweite Suchmaschinen müssen viele verschiedenartige Informationsquellen eines einzelnen Betriebes — wie Intranet, Dokument Management Systeme, Datenbanken und Dateiserver — nutzen und dabei auch unternehmensspezifisches Wissen mit einbeziehen. [CMS10]

Intranet-Suchmaschinen fokussieren sich auf nicht öffentliche Netzwerke beziehungsweise Teilnetze für geschlossene Benutzergruppen, wie zum Beispiel das Intranet eines Unternehmens.

Desktop-Suchmaschinen nutzen den permanenten Speicher eines einzelnen Computers als Datenbasis. Da die Speicherkapazität von Festplatten rasant ansteigt und immer mehr Anwender vermehrt digitale Medien nutzen, sind integrierte Suchmechanismen des Betriebssystems oder eMail-Programms oft nicht mehr ausreichend. Allmählich halten daher Suchmaschinen auch Einzug in den privaten PC. [Col05]

Vertikale Suchmaschinen verwenden nur einen Teil der Dokumente aus der Datenbasis. Als „vertikal“ werden sie bezeichnet weil sie nur Dokumente zu einem bestimmten Thema indexieren. [MRS08]

Echtzeit-Suchmaschinen machen es sich zur Aufgabe Informationen praktisch sofort zum Zeitpunkt ihrer Veröffentlichung suchbar zu machen. Anwendungsbereiche hierfür sind beispielsweise Blogs oder soziale Netzwerke.

2.1.3 Klassifizierung nach Dokumenttypen

Text, Bilder, Videos oder andere Dokumente?

Die Dateitypen der zu durchsuchenden Dokumente können ebenfalls zur Klassifizierung herangezogen werden. Obwohl wir in einer multimedialen Welt leben und moderne Suchmaschinen verschiedenen Medien wie Bildern, Audio-Dateien, Videos etc. Rechnung tragen müssen, kommt der textbasierten Suche eine spezielle Bedeutung zu.

Auch für nicht-textuelle Inhalte wie Videos oder Audio-Dateien werden heutzutage eher textuelle Beschreibungen der Inhalte — die *Metadaten*[1] — zur Klassifizierung und Suche herangezogen, als ihre tatsächlichen Inhalte. Obwohl beispielsweise beim direkten inhaltlichen Bildvergleich technische Fortschritte erzielt werden, stützt sich die praktische Implementation von Suchmaschinen auch bei nicht-textuellen Inhalten derzeit noch weitestgehend auf deren textuelle Beschreibungen. [CMS10]

2.2 Anforderungen

> „Eine gute Suchmaschine muss besonders rasch bestmögliche Resultate liefern!“

Dieser salopp formulierte Satz birgt die zwei wesentlichsten Anforderungen in sich. Hinter *„besonders rasch“* versteckt sich das Bestreben nach Performanz, sprich: Geschwindigkeit. Damit verbunden sind quantifizierbare Maßzahlen wie Antwortzeit und Durchsatz, die ihrerseits wiederum von unzähligen Faktoren abhängen.

Für *„bestmögliche Resultate“* sind die Qualität der gereihten Resultate und letztendlich auch die Berücksichtigung der individuellen Einschätzung durch den Benutzer sowie die Aktualität der Ergebnisse[2] ausschlaggebend.

Croft & Co [CMS10] titulieren diese beiden Hauptanforderungen nach Geschwindigkeit und Qualität als Bestreben nach *Effizienz* und *Effektivität*. Diese Betrachtungsweise ist in der Literatur zum Thema Suchmaschinen allgemein gebräuchlich und sei auch an dieser Stelle aufgegriffen.

2.2.1 Effizienz: Performanz, Geschwindigkeit

Benutzeranfragen sollen so schnell wie möglich abgearbeitet werden. Nachfolgend werden einige Maßzahlen für die Effizienz von Suchmaschinen erläutert. Diese Liste soll die wichtigsten Kriterien widerspiegeln und erhebt keinen Anspruch auf Vollständigkeit.

1 Metadaten sind „Daten *über* Daten“, wie beispielsweise Autor und Herausgeber eines Buches, das Motiv eines Bildes oder eine Szenebeschreibung eines Videos, während die tatsächliche Information *im* Buch, Bild oder Video enthalten ist.

2 und damit einhergehend die Aktualität des Index der Suchmaschine

Kennzahlen der Effizienz

Die Antwortzeit beschreibt die Zeitspanne zwischen dem Absenden der Anfrage durch den Benutzer und dem Erscheinen der Antwort. Im Kontext der Suchmaschinen hängt die Antwortzeit also von der Übertragungsrate des Netzwerks, der Aufbereitungsdauer der Suchanfrage, der Zeit für die eigentliche Suche sowie der Dauer für die Ergebnissortierung und die Bereitstellung der Resultate ab.[1] Die Antwortzeit muss minimiert werden um der Forderung nach Effizienz gerecht zu werden.

Der Durchsatz gibt an, wie viele Anfragen pro Sekunde abgearbeitet werden können. Wesentliche Einflussfaktoren hierfür sind die implementierten Algorithmen und auch die eingesetzte Hardware. Für gute Performanz ist also größtmöglicher Durchsatz gefordert.

Die Größe des Index gibt den Speicherplatzbedarf für den Index einer bestimmten Dokument-Kollektion[2] an. Aus Effizienzgründen und Kostensicht soll die Index-Größe möglichst klein gehalten werden.

Die Indexierungsdauer beschreibt wie lange ein System für die Indexierung eines bestimmten Dokumentsatzes[2] benötigt. Alternativ dazu kann auch die benötigte Prozessor-Zeit herangezogen werden. In diesem Fall werden periphere Wartezeiten für Ein/Ausgabe und Geschwindigkeitsgewinne durch Parallelisierung nicht gemessen. [CMS10] Je kürzer die Indexierungsdauer ist umso aktueller kann ein Index gehalten werden.

2.2.2 Effektivität: Relevanz, Qualität

Als Resultat der Suche sind die relevanten Dokumente zur gestellten Anfrage in adäquater Reihung erwünscht. Die *Relevanz* der Dokumente spiegelt die Antwort auf die Frage „Wie viel der gesuchten Information beinhaltet das Ergebnis?“ wider. Für jede einzelne Person stellt sich jedoch noch eine zweite Frage: „ Ist das Ergebnis für *mich* interessant?“. Hier spricht man von der *Pertinenz* des Ergebnisses.

Relevanz und Pertinenz

Ein Ergebnis kann zwar im Bezug auf die Fragestellung allgemein relevant sein weil es gesuchte Informationen beinhaltet, für einen bestimmten Benutzer jedoch trotzdem nicht pertinent erscheinen. Möglicherweise ist es in einer Fremdsprache verfasst oder dem Anwender der Text bereits bekannt.

1 Die Antwortzeit endet definitionsgemäß mit dem Erscheinen des ersten Zeichens des Ergebnisses beim Benutzer. Daher hat die Datenmenge des zu übertragenden Resultates keinen Einfluss darauf.

2 einige Beispiele sind in Abschnitt 2.6.1 angeführt

Gerade bei proprietären Suchmaschinen werden die Ergebnisse zunehmend personalisiert um auf diese Art neben der Relevanz auch die individuelle Pertinenz in die Ergebnisreihung einfließen zu lassen. Stalder und Mayer sprechen in diesem Zusammenhang von einem *zweiten Index*. Der geneigte Leser sei hier auf [BS09, Abschnitt „Der zweite Index“] verwiesen.

In jedem Fall ist die Aktualität der Ergebnisse mitentscheidend für die Qualität. Was hilft ein Resultat, das zwar im Index der Suchmaschine gefunden wird, dessen tatsächliches Dokument aber längst nicht mehr verfügbar ist? Die Größe und Wachstumsrate des *World Wide Web*, sowie die Tatsache dass immer mehr Inhalte dynamisch generiert werden, stellen vor allem Internet-Suchmaschinen vor eine immense Herausforderung. Um in einer möglichst guten Abbildung der gesamten Dokumentkollektion zu suchen, muss der Index nicht nur erstellt sondern auch permanent aktualisiert werden. [EE10]

Ziel der Suche ist es, möglichst alle relevanten Dokumente aus der gesamten Datenbasis abzurufen und gleichzeitig die Anzahl der abgerufenen aber irrelevanten Dokumente zu minimieren. [CMS10]

Das bedeutet die Schnittmenge aus relevanten und abgerufenen Dokumenten zu maximieren und gleichzeitig die Gesamtmenge der Abrufe zu minimieren. Im Idealfall sind die Mengen der relevanten und der abgerufenen Dokumente deckungsgleich. Abbildung 2.1 veranschaulicht diesen Zusammenhang.

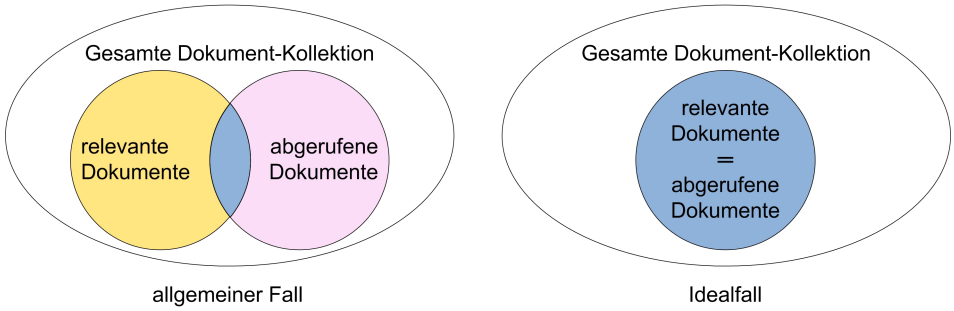

Abbildung 2.1: *Dokument-Mengen — Vergleich der Schnittmengen von abgerufenen und relevanten Dokumenten der Kollektion im Allgemein- beziehungsweise Idealfall*

Die wichtigsten Maßzahlen für die Effektivität sind *Präzision und Ausbeute*. Präzision und Ausbeute

Die Präzision gibt Aufschluss darüber, wie hoch die Rate der relevanten Dokumente unter den abgerufenen Ergebnissen ist. Je weniger irrelevante Dokumente unter den abgerufenen Ergebnissen sind, um so besser ist die Präzision. Sie soll selbstverständlich möglichst hoch sein.

Die Ausbeute wiederum besagt, wie hoch die Rate der abgerufenen Ergebnisse unter den relevanten Dokumenten ist. Je geringer die Menge der Dokumente ist, die zwar relevant sind aber dennoch nicht abgerufen wurden, um so höher ist die Ausbeute. Sie soll demnach ebenfalls möglichst groß sein.

Sind beide Werte optimal, so werden *alle relevanten Dokumente* abgerufen und gleichzeitig *kein einziges nicht relevantes Dokument* als Ergebnis geliefert. Das entspricht dem theoretischen Idealfall — den deckungsgleichen Mengen.

In der Praxis gibt es jedoch zwei Gruppen von Fehlern:

Falsch negative Ergebnisse sind Dokumente, die zwar relevant sind aber dennoch nicht abgerufen werden.

Falsch positive Ergebnisse sind Dokumente, die abgerufen werden obwohl sie nicht relevant sind.

Verständlicherweise sollen diese beiden Mengen möglichst gering sein.

2.2.3 Aufwand

Ergänzend zu den beiden voran stehenden Anforderungen sei noch der Aspekt des Aufwandes erwähnt.

Bei proprietären Suchmaschinen bedeutet Aufwand gleichermaßen anfallende Kosten und es gilt hier den Spagat zwischen diesen drei Punkten zu schaffen. Legt man die Priorität auf zwei dieser Faktoren, so resultiert daraus der Dritte: Wenn man hohe Ansprüche an Effektivität und Effizienz stellt, so resultiert daraus ein hoher Aufwand. Will man hingegen hohe Effizienz bei niedrigen Kosten erreichen, so leidet zwangsläufig die Effektivität darunter. [CMS10]

Es ist also offensichtlich dass man nicht gleichzeitig beliebig an diesen drei Schrauben drehen kann. Eine — unter allen drei Aspekten — optimale Realisierung käme der Quadratur des Kreises gleich. Man muss demzufolge eine Priorisierung vornehmen und den bestmöglichen Kompromiss finden. Der finanzielle Gesichtspunkt ist jedoch nicht Gegenstand dieser Arbeit und der Aufwand sei daher nicht eingehender betrachtet.

2.3 Architektur

Im letzten Abschnitt wurden die beiden Hauptanforderungen an Suchmaschinen — Effizienz und Effektivität — erläutert. Um diesen Ansprüchen gerecht zu werden, muss die Architektur moderner Suchmaschinen die beiden wesentlichen Prozesse der Indexierung und Suche bestmöglich unterstützen. So vielfältig wie die Einsatzgebiete und daraus resultierenden Anforderungen sind, so unterschiedlich sind auch die dafür verwendeten Architekturen. Eine Veranschaulichung des Aufbaus einer Suchmaschine ist daher nur sinnig wenn sie auf einer speziellen Implementation beruht oder bis zur Allgemeingültigkeit vereinfacht ist.

Abbildung 2.2 illustriert eine allgemeingültige Architektur von Suchmaschinen. Die beiden Prozesse der Indexierung und Suche sind horizontal von einander abgegrenzt. Die vertikale Teilung versucht die Suchmaschine physikalisch von ihrem Umfeld abzusondern und verdeutlicht, dass alle wesentlichen Komponenten — einschließlich der Crawler[1] — auf der Hardware des Suchmaschinenbetreibers laufen.

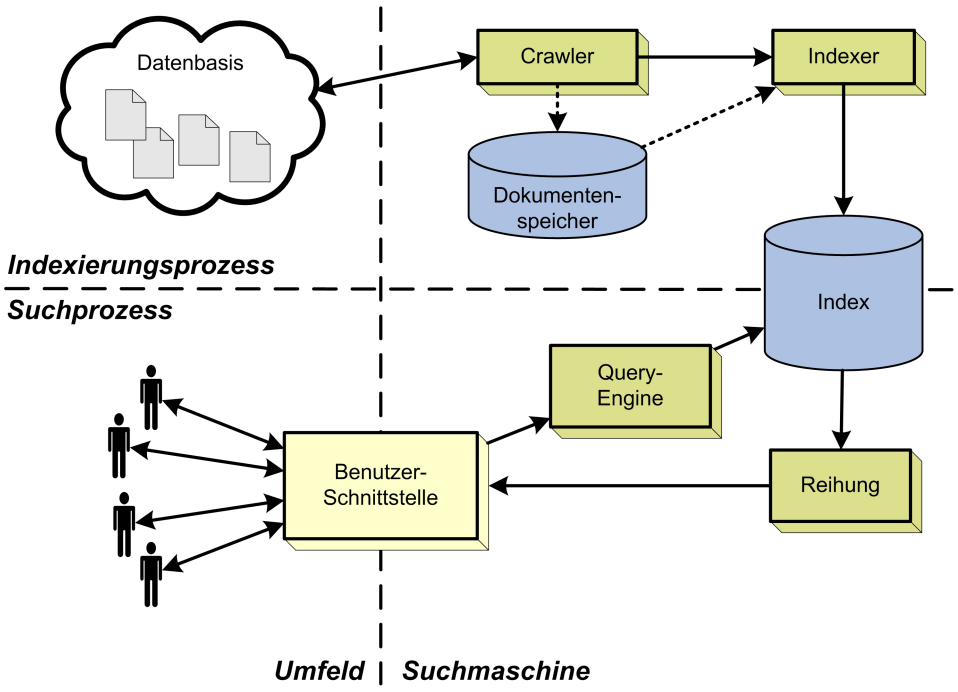

Abbildung 2.2: *Minimalarchitektur von Suchmaschinen*

Während der Indexierung sammelt ein *Crawler* die Informationen aus der Datenbasis und legt in der Regel eine Kopie in einem lokalen Dokumentenspei-

1 siehe Seite 19

cher ab. Der *Indexer* bereitet diese Informationen auf und erstellt daraus den Index, in dem die Suche tatsächlich vollzogen wird. Dazu wird die Anfrage des Benutzers über eine Schnittstelle an das System übergeben. Die sogenannte *Query-Engine* bereitet die eingegebenen Suchbegriffe — den *Query* — für die Abfrage am Index auf. Die gelieferten Ergebnisse werden nach Relevanz gereiht und wiederum über die Benutzeroberfläche dem Anwender präsentiert.

Auch aktuelle Internet-Suchmaschinen spiegeln diese wesentlichen architektonischen Bestandteile wider. Die Unterschiede gegenüber der vereinfachten Darstellung in Abbildung 2.2 liegen großteils darin, dass viele der Komponenten bzw. Prozesse parallelisiert werden. Durch den gleichzeitigen Einsatz ganzer Schwärme von Crawlern, der Verteilung des Index auf mehrere lokal getrennte Server oder Rechner-Farmen, sowie die lokale Speicherung von homogenisierten und komprimierten Abbildern der Basisdokumente bewirken notwendige Anpassungen des Aufbaus. Diese Adaptierungen sind von Implementation zu Implementation unterschiedlich. Auf einige dieser Abweichungen wird in Abschnitt 2.7 hingewiesen.

Nachfolgend sind die Prozesse der Indexierung und Suche in ihre essenziellen Bestandteile zerlegt und eingehender beschrieben. Das Hauptaugenmerk dieser Arbeit liegt auf textbasierten Suchmaschinen und folglich sind auch die nachstehenden Abschnitte dahingehend ausgerichtet.

2.4 Indexierung

Der Indexierungsprozess bleibt dem Benutzer in der Regel verborgen. Sein Zweck ist die Sammlung von Informationen aus der Datenbasis, deren Homogenisierung und Verdichtung. Das Endergebnis ist der *Index*, der die verdichteten Informationen und Referenzen zu den enthaltenden Basisdokumenten beinhaltet. Die Verweise zu den Ursprungsdaten können dabei auch mit einer Gewichtung behaftet sein um die Reihung der Ergebnisse im Rahmen des Suchprozesses zu verbessern. Der Index sei vorerst als allgemeine Datenstruktur verstanden, deren Ziel es ist die Suche unter Betracht der Anforderungen aus Abschnitt 2.2 bestmöglich zu unterstützen. Abbildung 2.3 veranschaulicht den Ablauf der Indexierung und dessen wichtigste Teilprozesse.

2.4.1 Text-Akquise

Das Geheimnis guter Suchmaschinen beginnt bereits bei der Akquise der Basisdokumente. Denn nicht indexierte Dokumente können auch unter Einsatz der

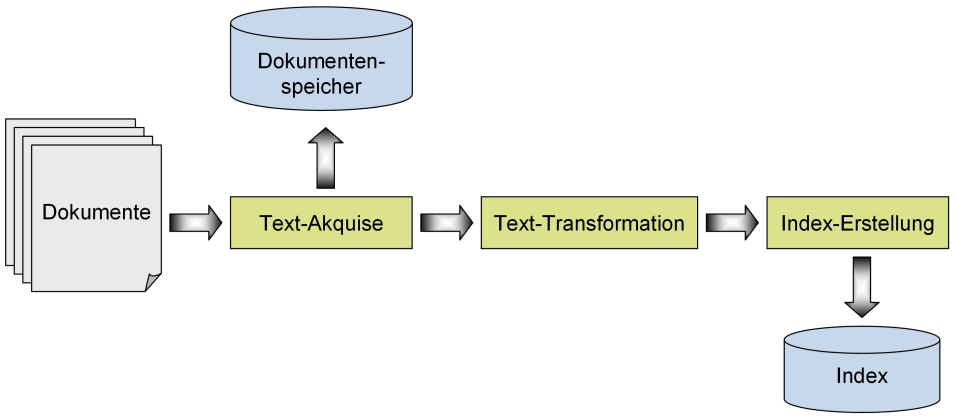

Abbildung 2.3: *Der Indexierungsprozess*
modifizierte Darstellung von [CMS10, Fig.2.1]

besten Technologien nicht gefunden werden, da die Suche stets im Index und nicht in den Ursprungsdokumenten durchgeführt wird. Die Text-Akquise muss also die Antwort auf die Frage *„Was soll eigentlich durchsucht werden?“* technisch realisieren. [CMS10]

Im einfachsten Fall ist die Antwort auf diese Frage bekannt, definiert und endlich. Beispiele hierfür sind etwa „die Stammverzeichnisse der unternehmenseigenen Web-Server“ für eine Intranet-Suchmaschine, „die lokalen Festplatten“ oder gar nur „alle Text-Dateien in einem spezifischen Ordner“ für eine Desktop-Suchmaschine. Für diese Anwendungsbereiche ist in der Regel ein Blick in das Dateiverzeichnis des Betriebssystems ausreichend um alle Dateien der Datenbasis zu lokalisieren.

Für eine allgemeine Beschreibung der Text-Akquise erscheint die komplexere Problemstellung bei Internet-Suchmaschinen als besser geeignet. Hier ist die Datenbasis auf unzählige Datenträger verteilt und es existiert kein all umgreifendes „Inhaltsverzeichnis“ dafür. Es ist offensichtlich, dass sich diese Aufgabe in der Praxis nur maschinell durch eigenständige Programme lösen lässt.

Agents, Ants, Crawler, Spider, Robots und Co

Obwohl die Bezeichnungen für solche Programme vielfältig sind, ist deren Zweck jedoch stets der selbe: So effizient wie möglich denkbar viele Dokumente der Basis lokalisieren, in den Festspeicher des Suchmaschinenbetreibers kopieren und diesen tunlichst auf aktuellem Stand halten. Die meisten Spider speichern neben den Dokumenten auch deren Verknüpfungsstruktur ab um sie in

die Ergebnisreihung mit einzubeziehen.[1] [MRS08] Da diese Programme dieselben Aufgaben erledigen seien auch ihre Bezeichnungen synonym verwendet.

Der Crawling Prozess

Den Ausgangspunkt für einen Crawler bildet eine definierte Liste von Quelldokumenten — die Start-URLs. Für alle Einträge in dieser Liste vollführt das Programm autonom folgende Schritte:

- Das Quelldokument wird mittels des Befehls *Get* aus dem *HyperText Transmission Protocol (HTTP)* bezogen. Zur Adressierung wird dabei der *Uniform Resource Locator (URL)* verwendet.
- Anschließend werden die im Dokument enthaltenen Verknüpfungen extrahiert und diese neuen Adressen mit einer bestimmten Methodik in die Liste der zu besuchenden Seiten aufgenommen.
- Das aktuelle Objekt wird indexiert und gegebenenfalls eine Kopie im Dokumentenspeicher der Suchmaschine abgelegt.
- Der Ablauf wird mit dem nächsten Eintrag aus der Liste wiederholt, bis die Liste abgearbeitet oder ein anderes Abbruchkriterium erfüllt ist.

[MMG+11], [CMS10]

Die Start-URLs legen also den Ausgangspunkt für die weitestgehend autonomen Crawler fest und haben somit wesentlichen Einfluss darauf, welche Dokumente indexiert werden. Das spiegelt sich letztendlich auch in der Qualität der Suchergebnisse wieder. Ein Crawler, der von verwalteten Seiten zu verschiedenartigsten Themen ausgeht[2], wird andere Dokumente aufspüren als wenn er Verknüpfungen ausgehend von einer Seite aus einem sozialen Netzwerk folgt. [Sla10]

Um die Wartezeiten während Namensauflösung[3], Verbindungsaufbau und Download des Dokumentes zu nutzen und somit die Effizienz zu steigern, werden Spider mit mehreren Threads[4] realisiert. Auf diese Art können hunderte Dokumente gleichzeitig von einem einzelnen Search-bot bezogen werden. [CMS10]

Multi-Threading und Parallelisierung

Moderne Internet-Suchmaschinen gehen hier noch bedeutend weiter. Viele Spider (von denen jeder hunderte Dokumente gleichzeitig abruft) durchforsten simultan eine gemeinsame Datenbasis. Dadurch ergeben sich wiederum Heraus-

1 Gegenwärtig stellt die *Link-Analyse* das gängigste Verfahren zur Beurteilung der Relevanz eines Dokumentes für Internet-Suchmaschinen dar, siehe Seite 28

2 beispielsweise das *Open Directory Project*, www.dmoz.org

3 Bei der Namensauflösung — dem *DNS lookup* — wird die menschenlesbare URL vom Domain Name Service (DNS) in die numerische *IP-Adresse* umgesetzt

4 http://de.wikipedia.org/wiki/Thread_(Informatik)

forderungen in Puncto Überlappung[1] und Kommunikation[2]. [MMG+11] Ein tieferer Einblick in diese Thematik des parallelen Crawlings würde den Rahmen dieser Arbeit sprengen.

Erwähnenswert ist, dass Robots einigen Studien zur Folge ein Drittel des gesamten Datenverkehrs im Internet verursachen! [YMH02] Sie sind also bei der Dimensionierung von Web-Servern eine beachtenswerte Einflussgröße geworden. Das *Robots exclusion Protocol*[3] bietet hier Seitenbetreibern die Möglichkeit das Verhalten von Robots zu beeinflussen um eine Überlastung ihrer Server durch maschinell verursachte Anfragen zu vermeiden. [DG11] Man spricht in diesem Zusammenhang vom „ethnischen Verhalten“ oder der „Höflichkeit“ von Robots. Ein alternativer Ansatz wäre das Crawling durch Monitoring an strategischen Routern im Web zu ersetzen und so die benötigten Daten zu ermitteln. [SCG10]

Datenverkehr

Ergänzend sei noch bemerkt, dass auch im eingangs erwähnten „einfachen Fall“ von Desktop-Suchmaschinen die Text-Akquise durch sogenannte *File-Crawler* geschieht. Sie folgen, ähnlich wie ihre Verwandten im Internet, Verknüpfungen zwischen Dokumenten und/oder durchsuchen Verzeichnisse der lokalen Festspeicher. [Col05]

Feeds

Viele Dokumente werden erstellt, veröffentlicht und in weiterer Folge kaum verändert. Anhand des Datums lassen sich solche Dokumente von einer einzelnen Quelle zu einer Reihe sortieren. Die Publikationen eines Verlages, die Ausgaben eines Journals oder die Einträge in einem Blog stellen solche Reihen dar. In der informationstechnischen Fachsprache werden sie als *Feeds* bezeichnet. [CMS10]

Ihnen kommt bei der Akquise eine spezielle Bedeutung zu, weil Crawler nur das Ende dieser Reihen betrachten müssen um neue Dokumente zu finden. Bei technischen Implementationen wie dem *RSS Feed*[4] stehen dafür auch Methoden wie *Monitoring* oder *aktive Notifikation* zur Verfügung. Ein weiterer Vorteil von RSS-Feeds liegt darin, dass sie als XML-Format definiert sind und somit sehr einheitliche Objekte darstellen. Zusätzlich sind Zeitstempel und Lebensdauer der Informationen enthalten, die es erleichtern den Index aktuell zu halten.

RSS-Feed

1 Mehrere Crawler können in ihren Dokumenten Verknüpfungen zur selben URL finden
2 um diese Überlappungen zu behandeln
3 in Anlehnung an die zentrale Datei auch als *Robots.txt Protocol* bekannt
4 RSS steht für „Really Simple Syndication“, „RDF Site Summary“ oder auch „Rich Site Summary“

Echtzeit-Suchmaschinen machen starken Gebrauch von Feeds um damit Informationen bereits im Moment ihres Entstehens aufzuspüren und zu indexieren. [SLP11]

Konvertierung

Desktop-Suchmaschinen müssen mit verschiedenen Objekten — wie HTML-Seiten, eMail-Dateien, Datenbanken, Office-Dokumenten und vielen mehr — zurande kommen. Das trifft in ähnlicher Form auch auf textbasierte Internet-Suchmaschinen zu. [Colo5]

heterogene Datenbasis

Quelltexte können in unterschiedlichen Formaten wie HTML[1], XML[2], DOC[3], RTF[4] oder Hunderten anderen Textformaten vorliegen. Aber auch andere Dokumenttypen wie PDF[5], PPT[6] oder XLS[7] können bei der Indexierung oftmals nicht einfach ignoriert werden. Für kommerzielle Suchmaschinen ist es daher nicht unüblich über hundert verschiedene Dateitypen heranzuziehen.

Man kann sich gut vorstellen, dass es zur Konvertierung von hundert divergenten Dateitypen in ein einzelnes, weiterverarbeitbares Format einigen Aufwandes und Gehirnschmalzes bedarf. Tatsächlich ist aber schon vor dieser Konvertierung ein anderes, nahezu unlösbar erscheinendes Problem zu bewältigen, die *Bestimmung der Kodierung*. [Sko10]

Kodierung

Aufgrund verschiedener Sprachen und Schriftzeichen haben sich viele verschiedene Standards entwickelt um Zeichen digital zu kodieren. Das vom Crawler abgerufene Dokument ist vorerst nur ein Byte-Strom. Dieser Byte-Strom muss korrekt interpretiert werden um an den enthaltenen Text zu gelangen. Viele Dateien enthalten dazu entweder in ihrer Präambel oder im Inhalt Angaben zur verwendeten Kodierung. Um diese Angaben jedoch korrekt auslesen zu können muss aber wiederum die Kodierung bereits bekannt sein...

Unter dem Strich bleibt hier laut Wikipedia nur die Möglichkeit mittels heuristischer Methoden und Mustererkennung die Kodierung zu *„erraten“*, um dann den Text extrahieren und konvertieren zu können.

1 „HyperText Markup Language“, Auszeichnungssprache und Grundlage des *World Wide Web*
2 „eXtensible Markup Language“, erweiterbare Auszeichnungssprache
3 Dateiformat von Microsoft Word
4 „Rich Text Format“, von Microsoft entwickeltes Dateiformat mit Elementen zur Steuerung des Layouts
5 „Portable Document Format“, von Adobe zum Plattform-unabhängigen Austausch von Dokumenten entwickelt
6 Dateiformat von Microsoft PowerPoint
7 Microsoft Excel Arbeitsmappe

Sammlung im Dokumentenspeicher

Während des Vorgangs zur Index-Erstellung werden oft auch homogene Kopien der Quelldokumente von der Suchmaschine gespeichert. Diese lokalen und meist komprimierten Daten können in weiterer Folge sowohl für Vorschauen bei der Ergebnispräsentation[1] verwendet werden, als auch die Basis für den Indexer bilden. Ein weiterer Vorteil liegt darin, dass bereits gespeicherte Dokumente bei einer Aktualisierung des Index nicht erneut abgerufen werden müssen. Mittels *HEAD-Request* und Vergleich mit den Metadaten des lokalen Abbildes kann festgestellt werden ob das Basisdokument geändert wurde. Nur neue oder veränderte Dokumente werden abgerufen und somit Zeit und Datenverkehr gespart. Bei Desktop-Suchmaschinen wird auf einen eigenen Dokumentenspeicher meist verzichtet, da die Basisdokumente ohnehin im Dateisystem der Suchmaschine greifbar sind. [CMS10]

Üblicherweise werden die abgerufenen Dokumente vor dem Speichern in eine Auszeichnungssprache übersetzt. HTML oder XML bieten sich hierfür an da sie einerseits die Markierung von Metadaten erlauben und andererseits den Inhalt auch allgemein zugänglich machen.[2]

2.4.2 Text-Transformation

Die Text-Transformation schließt an die Akquise an und stellt nach Abbildung 2.3 den nächsten Teilprozess der Indexierung dar. Ihr Ziel ist die Überführung der verschiedenen Wortformen auf einheitliche *Terme* um die Trefferwahrscheinlichkeit zu erhöhen. Eine exakte Suche innerhalb eines Textes — wie beispielsweise die Such-Funktion in Text-Editoren — ist recht restriktiv. Bereits eine unterschiedliche Groß-/ Kleinschreibung zwischen Suchbegriff und dessen Erscheinung im Dokument würde ein Auffinden verhindern. Ein erster Ansatz zur Konvertierung ist demnach die Groß-/ Kleinschreibung zu ignorieren indem man den gesamten Text an einer geeigneter Stelle des Prozesses auf eine einheitliche Schreibweise überführt. [CMS10]

Wie die nachfolgenden Abschnitte zeigen, gehen moderne Suchmaschinen bei der Text-Transformation bedeutend weiter. Vorab seien jedoch einige Begriffe nach Manning & Co [MRS08] definiert:

1 sogenannten Snippets
2 Benutzer können beispielsweise die HTML-Version eines PDF's aus dem Dokumentenspeicher der Suchmaschine ansehen ohne einen PDF-Reader installiert zu haben. Das ist besonders bei obsoleten Dateiformaten von Vorteil.

Ein Token ist eine Gruppe von Zeichen, die eine semantisch nützliche Einheit zur weiteren Verarbeitung darstellt.

Ein Typ ist die Klasse aller Token mit der selben Zeichensequenz.

Ein Term ist ein gegebenenfalls normalisierter Typ, der in den Index aufgenommen wird.

Die Gesamtheit der Terme bildet das *Vokabular* des Index.

Textzerlegung — Tokenizing

Unter Textzerlegung versteht man die Auflösung eines Textes in seine „Bausteine“, die *Token*. Typischerweise stellt ein einzelnes Wort einen solchen Token dar. Aber auch eine gesamte Phrase oder eine eMail-Adresse können einen Baustein repräsentieren. Die Identifikation der Token kann dabei auf zwei Arten erfolgen:

- Zerlegung des Textes in Sätze und anschließende Aufspaltung der Sätze in ihre Bestandteile
- „Aufsummieren“ von einzelnen Zeichen bis zu einem Trennzeichen

Der erste Ansatz stellt eine Top-down Methodik dar. Das Hauptproblem dabei ist, dass Quelltexte oft nicht in Sätze zerlegt werden können.[1] Die zweite Herangehensweise nach dem Bottom-up Prinzip entgeht diesem Dilemma und ist darüber hinaus auch effizienter implementierbar. [Kono8]

Selektion der Trennzeichen

Die Wahl der Trennzeichen zur Abgrenzung der Token ist sprachabhängig und nicht trivial. Ein Leerzeichen muss nicht zwangsläufig das Ende eines Tokens kennzeichnen. *„Los Angeles“* sollte semantisch als eine Einheit verstanden werden anstatt geteilt zu werden. In der deutschen Sprache ist bei zusammengesetzten Hauptwörtern oft das Gegenteil der Fall. Sie beinhalten keine Leerzeichen und sollten eventuell dennoch als getrennte Token betrachtet werden. Ähnlich verhält es sich mit Apostrophen, die beispielsweise in englischer und französischer Sprache unterschiedlich behandelt werden sollten.

Auch Satzzeichen sind nicht immer zuverlässige Trennzeichen. In Abkürzungen oder eMail-Adressen stellt der Punkt einen Teil des Tokens dar anstatt ihn zu begrenzen. In der Praxis gibt es viele verschiedene Ansätze um diesen Problemen beizukommen. Der Einsatz von Programmen zur Spracherkennung um

1 Man denke zum Beispiel an PowerPoint-Präsentationen, die meist nur aus Phrasen bestehen und nahezu keine vollständigen Sätze beinhalten.

den richtigen Satz von Trennzeichen zu bestimmen, die zusätzliche Verwendung von Listen mit Zeichenfolgen die als einzelner Token betrachtet werden sollen oder Methoden zur Trennung zusammengesetzter Wörter sind einige Beispiele hierfür. Auch eine Erweiterung der Suchbegriffe, die sogenannte *Query-Expansion*, um mehrere bedeutungsgleiche oder ähnliche Terme im Index zu finden, stellt eine mögliche Abhilfe dar. [MRS08]

Es gibt also keine allgemeingültige Definition wie Textzerlegung stattzufinden hat. Auch spezielle Anforderungen wie die Sprache der Quelldokumente können hierauf Einfluss haben. [Kon08]

Textbereinigung — Stoppwörter

Die natürliche Sprache ist gespickt mit Wörtern die von geringer Aussagekraft für den Inhalt sind. Fürwörter, Artikel und Bindewörter machen einen Gutteil von Texten aus, sind aber für die inhaltliche Beschreibung und damit zur Suche praktisch ungeeignet. Tabelle 2.1 listet die dreißig häufigsten Wörter in deutschen Texten auf.[1] Ein Blick auf diese Tabelle verdeutlicht intuitiv, dass die häufigsten Wörter nur wenig Bedeutung für eine Suche haben. Sie wären faktisch nur Ballast und es liegt daher nahe sie nicht als Terme in den Index aufzunehmen. Zu diesem Zweck werden oft jene Token des Quelltextes verworfen, die sich auf einer *Stoppwortliste* befinden.

Rang	Wort	Rang	Wort	Rang	Wort
1	der	11	des	21	als
2	die	12	auf	22	auch
3	und	13	für	23	es
4	in	14	ist	24	an
5	den	15	im	25	werden
6	von	16	dem	26	aus
7	zu	17	nicht	27	er
8	das	18	ein	28	hat
9	mit	19	die	29	dass
10	sich	20	eine	30	sie

Tabelle 2.1: *Die 30 häufigsten Wörter in deutschen Texten*
Datenquelle: Abteilung Sprachverarbeitung der Universität Leipzig,
http://wortschatz.uni-leipzig.de/html/wliste.html

1 Solche Statistiken sind naturgemäß von Anzahl und Art der untersuchten Quelldokumente abhängig. Tabelle 2.1 ist daher als beispielhaft anzusehen.

Die Verwendung vieler Stoppwörter kann sich negativ auf die Ausbeute auswirken!

Die Stoppwortliste vermindert einerseits die Index-Größe, will aber andererseits sorgfältig definiert sein. Eine Liste der x häufigsten englischen Wörter könnte beispielsweise dazu führen, dass das Zitat *„To be or not to be“* aus Shakespeares Hamlet nicht indexiert wird, da es nur aus Stoppwörtern besteht. [CMS10] Das würde die Ausbeute[1] dramatisch beeinflussen.

Normalisierung der Token

Im besten Fall liefert ein Token aus der Suchanfrage einen direkten Treffer im Index. Die Menge der Treffer lässt sich durch eine Normalisierung der Token vergrößern. Diese hat sowohl bei der Indexierung als auch bei der Aufbereitung der Benutzeranfrage zu erfolgen.

Ein einfaches Verfahren dazu stellt die bereits erwähnte Überführung auf Kleinschreibweise dar. In einigen Sprachen wie Deutsch oder Englisch lassen sich Akzente und andere diakritische Zeichen[2] ohne wesentlichen Informationsverlust entfernen. In anderen Sprachen wie Spanisch kann ein diakritisches Zeichen hingegen der einzige Unterschied zwischen zwei gänzlich verschiedenen Wörtern sein. Abhängig von den gewählten Trennzeichen bei der Textzerlegung kann sich auch hier ein Feld für die Normalisierung auftun. Werden Apostrophe und Bindestriche beispielsweise nicht als Token-Grenzen verwendet, so kann man sie bei der Normalisierung verwenden um aus einem Token mehrere Terme zu generieren und diese Terme miteinander in Beziehung setzen. So entstehen *Äquivalenzklassen*, die bei der Suche ebenfalls mit einbezogen werden können. [MRS08]

Kanonisierung — Stemming vs. Lemmatisierung

Eine weitere Vergrößerung der Ergebnismenge lässt sich durch Normung der Token auf eine einheitliche Wortform erreichen. Dadurch werden zu einem Query auch Treffer erzielt, die den Suchbegriff in anderen Formen enthalten. Auf diese Weise kann eine Suche nach „Haus“ beispielsweise auch Verweise auf Quelldokumente liefern, die die Mehrzahl „Häuser“ enthalten. Gleiches gilt

1 siehe Seiten 16, 46
2 wie Hatschek, Tilde, Trema und Co

sinngemäß auch für verschiedene Zeitformen, Steigerungen und andere Wortarten. Die Kanonisierung kann auf zwei Arten erfolgen.

Stamm, Lemma

Algorithmen-basiert: Bei der Transformation mittels Algorithmen werden bekannte Affixe[1] in der Hoffnung abgeschnitten dadurch einen eindeutigen Wortstamm zu erhalten. Dieses Verfahren wird als *Stemming*. bezeichnet.

Wörterbuch-basiert: Bei dieser Methode wird ein Token unter Verwendung von Wörterbüchern und morphologischer Analyse auf einen Stamm übergeführt. Man spricht hier von *Lemmatisierung*.

[MRS08], [CMS10]

Beim Stemming wird der Token auf einen Stamm zurückgeführt, den es in der natürlichen Sprache nicht zwingend tatsächlich geben muss. Bei der Lemmatisierung hingegen wird der Token auf seine tatsächliche Wörterbuchform, das *Lemma*, transformiert. [BGH+06]

Progressives Stemming vergrößert die Menge der falsch-positiven Resultate. Stemming sollte daher tendenziell konservativ betrieben werden!

Das Algorithmen-basierte Stemming ist hinsichtlich der Indexierungsdauer günstiger. Da es den Token jedoch aufgrund von Heuristiken beschneidet, ergibt sich hier eine mögliche Fehlerquelle — semantisch unterschiedliche Wörter werden eventuell fälschlicherweise auf *einen* (fiktiven) Wortstamm reduziert und führen somit zu falsch-positiven Treffern.[2] Bei besonders progressiven Algorithmen treten solche Fehler vermehrt auf. Das exakte Verfahren mittels Wörterbüchern eliminiert diese Schwäche auf Kosten der Performanz. Eine Entscheidung für eines dieser Verfahren hängt vom Anwendungsfall ab. Neben der Performanz ist dabei auch die Komplexität der jeweiligen Sprache ein Entscheidungskriterium.

Analyse — Struktur, Verknüpfungen, Phrasen und Co

Die voran stehenden Abschnitte beziehen sich primär auf den Inhalt der Dokumente und was damit während der Indexierung geschieht. Es sieht bislang so aus, als ob nach der Text-Transformation nur eine Menge von Termen zur

1 vor allem Präfixe und Suffixe
2 siehe Seite 16

Erstellung des Index zur Verfügung steht. Die Reihung der Ergebnisse könnte folglich nur auf quantitativer Basis erfolgen, etwa nach der Anzahl der Worterscheinungen.

Ist ein Treffer im Titel eines Dokuments jedoch nicht bedeutsamer als ein Treffer irgendwo im Fließtext? Ist eine selten referenzierte Quelle wirklich gleich relevant wie ein Dokument, das von vielen anderen Objekten der Datenbasis verknüpft ist? Um solche Überlegungen bei der Reihung der Ergebnisse einbeziehen zu können werden zusätzliche Informationen benötigt, die in separaten Feldern im Index gespeichert werden. Welche Informationen dafür in Frage kommen, hängt vorrangig von der Dokumentkollektion und dem Anwendungsbereich der Suchmaschine ab. *Wie* man diese Daten nutzbar machen kann ist wiederum eine andere Geschichte. Dem Erfindungsgeist sind hierbei praktisch keine Grenzen gesetzt.

Struktur-Analyse

Nachstehendem Abriss einiger dieser zusätzlichen Daten und Methoden zur Extraktion liegt das Buch von Croft & Co [CMS10] zugrunde. Bei HTML-Dokumenten lassen sich ausgezeichnete Daten wie Titel oder Metadaten recht leicht anhand ihrer Markierungen bestimmen. Die Herausforderung ist es, auch für Quellen in anderen Formaten eine möglichst gute Strukturanalyse durchzuführen und semantisch wertvolle Daten für den Index bereitzustellen. Den *Hyperlinks* kommt hier eine spezielle Bedeutung zu.

Link-Analyse

Der Ankertext stellt oft eine kurze und prägnante Beschreibung des Inhalts der verknüpften Seite dar. Der Benutzer gibt bei der Suche ebenfalls eine kurze, prägnante Beschreibung des erhofften Inhaltes an. Ein Abgleich der Suchanfrage mit Ankertexten kann daher zu guten Resultaten führen. Darüber hinaus ist das Ziel eines Links stets eine URL. Aufgrund der Eindeutigkeit dieser URL lässt sich also leicht zählen wie oft auf eine Ressource verwiesen wird. Häufig referenzierte Dokumente werden im Allgemeinen als bedeutsamer ermessen und daher stärker gewichtet.

Phrasen und *n*-Gramme

Oftmals bestehen Suchanfragen aus mehreren Begriffen oder einer ganzen Phrase. Dokumente die diese Wörter in kurzem Abstand enthalten erscheinen besonders relevant. Dadurch ergibt sich die Überlegung Phrasen speziell zu behandeln. Dafür gibt es drei unterschiedliche Ansätze:

- Phrasen mittels Analyse der syntaktischen Struktur erkennen[1] und separat in den Index aufnehmen
- Die Position jedes Terms innerhalb des Quelldokuments speichern und während der Suche die Distanz zwischen den einzelnen Termen berechnen

1 zum Beispiel mittels *Part Of Speach (POS) Tagger*

- Generell Folgen von n Wörtern — sogenannte *n-Gramme* — bilden und diese separat in den Index aufnehmen

Die ersten beiden Verfahren haben wesentliche Nachteile in Hinsicht auf Indexierungsdauer und Antwortzeit. In der Praxis wird die Verwendung von n-Grammen bevorzugt, da sie einen guten Kompromiss zwischen dem negativen Einfluss auf die Index-Größe und der positiven Auswirkung auf das Suchergebnis darstellt. Auch der zusätzliche Zeitbedarf bei der Indexierung und der resultierende Qualitätsgewinn stehen für große Datenbasen in einem guten Verhältnis zueinander.

Im Rahmen der Dokument-Analyse gibt es noch viele weitere erwähnenswerte Konzepte wie die Bestimmung von benannten Entitäten.[1] Die vorgestellten Konzepte sollten jedoch genügen um dem Leser einen Eindruck zu vermitteln welch weitschichtiges Betätigungsfeld sich hier ergründet.

Alle genannten Methoden zur Text-Transformation haben ihre Stärken und Schwächen. Eines haben sie aber gemeinsam:

Es ist essenziell, dass sowohl bei der Indexierung als auch bei der Aufbereitung der Suchanfrage die selben Transformationen angewandt werden!

Würden hier verschiedene Stoppwörter oder Methoden zur Text-Zerlegung beziehungsweise Wortstammbildung zum Einsatz gebracht, so hätte das verheerende Auswirkungen auf das Ergebnis. Der Term im Index und der transformierte Suchbegriff könnten sich unterscheiden obwohl der Query exakt dem Wort im Quelltext entspricht. Schlechte Ergebnisse wären somit vorprogrammiert.

2.4.3 Index-Erstellung

Die zuvor erläuterten Konzepte beschreiben die Sammlung von Daten und deren Aufbereitung um daraus schlussendlich einen Index erstellen zu können, der eine effektive und effiziente Suche ermöglicht. Der vorliegende Abschnitt bringt diesen letzten Teilprozess der Indexierung aus Abbildung 2.3 näher.

Bislang wurde der Begriff „Index“ lediglich als Platzhalter für eine Datenstruktur verwendet, deren Ziel es ist die Suche hinsichtlich der Anforderungen aus

1 *„named entity recognition“*, Erkennung von Eigennamen, Ortsnamen, Datums- oder Währungsangaben…

Abschnitt 2.2 optimal zu unterstützen. Doch wie sieht diese Datenstruktur tatsächlich aus und wie wird sie befüllt?

Datenstruktur

Intuitiv möchte man vielleicht meinen, dass eine *relationale Datenbank* eine geeignete Struktur darstellt. Das mag für Einzelfälle zutreffend sein, im Allgemeinen stoßt man dabei nach Konchady [Kono8] aber auf folgenden Widerspruch: (relationale) Datenbanken sind geschaffen um *strukturierte* Daten zu verwalten, die Datenbasis für Suchmaschinen besteht in der Regel allerdings aus *unstrukturierten* Dokumenten. Diese Technologie ist daher für Suchmaschinen im Großen und Ganzen nicht praktikabel.

Suchmaschinen verkörpern einen Teilbereich des wissenschaftlichen Gebiets des *Information Retrieval* und implementieren zumeist den dort sehr verbreiteten *Invertierten Index*. Der „Invertierte Index" ist als Sammelbegriff für verschiedene Strukturen zu verstehen, die dem selben Grundgedanken folgen: Die bei der Text-Transformation ermittelten *Dokument* $\rightarrow$ *Term* Relationen werden zu *Term* $\rightarrow$ *Dokument* Relationen invertiert. [CMS10] Diese sind als Listen abbildbar und ihre Gesamtheit stellt den invertierten Index dar. Aufgrund dieser Tatsachen wird er auch oft als *Invertierte Datei* oder *Invertierte Liste* bezeichnet.

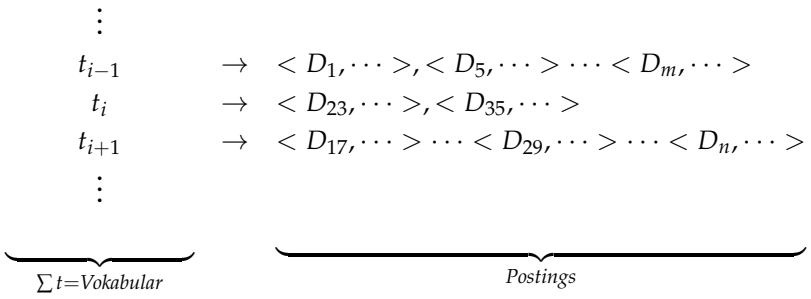

Abbildung 2.4: *Schematische Darstellung eines invertierten Index*
kombinierte Darstellung aus [Sino1, Abschnitt 2.4] und [MRSo8, Fig. 1.3]

Die Darstellung 2.4 stellt einen invertierten Index schematisiert dar. Die Summe der Terme t_j bildet das Vokabular des Index. Die Elemente in spitzen Klammern werden als *Postings* bezeichnet und stellen die Datensätze des Index dar. Dabei repräsentieren $D_1 \ldots D_n$ die indexierten Dokumente. Die Punkte innerhalb der Klammern deuten an, dass in der Regel weitere Informationen im Posting enthalten sind.

Bestandteile des Index

obligatorische Bestandteile

Die einzigen tatsächlich unerlässlichen Informationen in einem invertierten Index sind die Terme, die Adressen der indexierten Dokumente D_i[1] und die Relationen von Termen zu deren beinhaltenden Dokumenten. Ohne diese Information kann man von einem Suchbegriff nicht auf ein Dokument rückschließen. Stehen nur diese Informationen zur Verfügung, so ist keine Reihung der Ergebnisse hinsichtlich ihrer Relevanz möglich.

optionale Bestandteile

Um eine derartige Sortierung bewerkstelligen zu können muss das Gewicht eines Terms in einem Dokument bekannt oder bestimmbar sein — sprich: „Wie (ge)wichtig ist Term t_i im Dokument $D_m, D_n \ldots$?“. Dieses Gewicht kann anhand verschiedener Methoden bestimmt werden. Dabei wird oft auf folgende Größen zurückgegriffen.

Term-Frequenz *tf*: Je öfter ein Term in einem Dokument vorkommt, umso wichtiger erscheint er für dieses Dokument. In längeren Dokumenten kommen Terme aufgrund von Wortwiederholungen grundsätzlich öfter vor. Das muss aber nicht unbedingt bedeuten dass dieses Dokument relevanter ist als ein kürzerer Text mit weniger Worterscheinungen. Um den Einfluss der Länge zu mindern wird die Term-Frequenz meist anhand der Dokumentlänge normiert.

Dokument-Frequenz *df*: Terme die in vielen Dokumenten auftreten gelten als allgemein gebräuchlich und sind daher zur Indikation des Inhalts nur bedingt geeignet. Im Umkehrschluss sind selten auftretende Terme schwerwiegender und man rechnet daher mit der inversen Dokument-Frequenz *idf*

Diese Informationen sind im Index nahezu aller modernen Suchmaschinen zu finden. [Sino1]

Auch die Position eines Terms innerhalb des Quelltextes ist von Bedeutung. Bei Suchanfragen mit mehreren Wörtern ist der Abstand der Terme im Dokument für dessen Relevanz ausschlaggebend. Auch spezielle Dokumentteile wie Titel, Schlüsselwörter oder Überschriften werden bei der Ergebnisreihung zu Recht stärker berücksichtigt als „normale“ Wörter im Fließtext. Sie werden deshalb oft als separate Felder im Index angelegt. Wenn rechenintensive Algorithmen zur Bestimmung der Relevanz eingesetzt werden, können bereits während der Indexierung Dokument-spezifische Gewichte für die Terme dieser Felder berechnet werden. Sie können ebenfalls im Index abgelegt werden und somit Rechenzeit während der Suche sparen. [CMS10]

1 beziehungsweise Dokument-Identifikatoren, die *DocIDs*

Zu Beginn dieses Abschnitts wurde darauf hingewiesen, dass das zur Ergebnisreihung eingesetzte Verfahren wesentliche Auswirkungen auf die im Index benötigten Daten hat. Es wird daher von einer weiteren Aufzählung optionaler Bestandteile Abstand genommen und zur technischen Realisierung des Index übergesprungen.

Konstruktion des invertierten Index

Für ein leichteres Verständnis erfolgt eine schrittweise Annäherung von einer einfachen Methode zu einem komplexeren Verfahren. Die Schilderung der Index-Konstruktion basiert auf *„Introduction to Information Retrieval“* von Manning & Co. [MRS08, Kapitel 4]

Vorweg seien einige Fakten lose aufgelistet, die dem besseren Verständnis der folgenden Algorithmen dienen:

- Datenzugriff im Hauptspeicher ist wesentlich schneller als Zugriffe auf periphere Festspeicher.
- Lese-/ Schreibzugriffe auf Festplatten sind leistungsfähiger, wenn die Daten dort örtlich benachbart gespeichert sind.
- Während Lese-/ Schreibzugriffe vom *Bus* abgewickelt werden liegt Prozessorkapazität brach. Sie kann verwendet werden um die zu transferierende Datenmenge durch Kompression zu verringern. Dadurch lässt sich insgesamt die Performanz steigern.
- Die Architektur von modernen Suchmaschinen tendiert zu Farmen von Standard-Computern und nicht zu wenigen Super-Computern.
- Je nach Datenbasis kann der Index größer sein als der Haupt- oder Festspeicher eines einzelnen Computers.
- Parallelisierung verkürzt die Indexierungsdauer und ermöglicht daher höhere Auffrischungsraten.

Blocked Sort-Based Indexing BSBI: Bei kleinen Datenbasen kann die Index-Erstellung gänzlich im Hauptspeicher stattfinden und erst das Endprodukt als Datei abgelegt werden. Für größere Dokumentkollektionen ist die zusätzliche Verwendung von Festspeichern nötig. Wie bereits erwähnt ist es dabei wichtig die Daten örtlich sequenziell im peripheren Speicher zu halten. BSBI in Abbildung 2.5 erfüllt diese Anforderung durch blockweisen Zugriff (4, 6).

```
BSBIndexConstruction()
1  n ← 0
2  while not allDocumentsProcessed
3  do n ← n + 1
4     block ← ParseNextBlock()
5     BSBI-Invert(block)
6     WriteBlockToDisk(block, f_n)
7  MergeBlocks(f_1, ..., f_n; f_merged)
```

Abbildung 2.5: *Pseudocode des Blocked Sort-Based Indexing Algorithmus*
Quelle: [MRS08, Fig. 4.2]

Der Algorithmus analysiert die eingelesenen Dokumente und baut im Hauptspeicher einen Block von Term-Dokumentrelationen mit definierter Größe auf (4). Präziser formuliert wird aus Effizienzgründen ein Block aus *Term-* und *Dokument-Identifikatoren* aufgebaut,[1] der den vorhandenen Arbeitsspeicher optimal nutzt. Zeile 5 beschreibt die zweistufige Invertierung des Index:

- Sortierung der Term-Dokument Paare nach der Term-ID
- Akkumulieren der Paare mit identischer Term-ID zu Posting-Listen

Der so erhaltene invertierte Index des aktuellen Blocks wird auf die Festplatte ausgelagert (6) und der nächste Durchlauf der While-Schleife beginnt. Nachdem alle Dokumente derart abgearbeitet wurden, werden die einzelnen Indexe zu einem gemeinsamen, invertierten Index verschmolzen (7).

Der Vorteil des BSBI-Algorithmus liegt in der guten Skalierbarkeit. Nachteilig ist hingegen, dass eine Datenstruktur zur Abbildung der Terme auf die Term-IDs nötig ist, die bei großen Datenbasen möglicherweise die Kapazität des Arbeitsspeichers überschreitet.

Single-Pass In-Memory Indexing SPIMI: SPIMI entgeht dem zuvor beschriebenen Problem des BSBI-Algorithmus indem es die ursprünglichen Terme nutzt (statt sie auf numerische Term-IDs überzuführen) und für jeden behandelten Block eine Index-Datei abspeichert. Dadurch lässt sich ein invertierter Index für Dokumentkollektionen beliebiger Größe verwirklichen.

1 Die Abbildung von Termen und Dokumenten auf deren numerische, eindeutige Identifikatoren kann während der Analyse oder in einem separaten, vorgelagerten Durchlauf passieren.

Ein in Abbildung 2.6 nicht dargestellter Teil des gesamten SPIMI-Algorithmus analysiert vorerst die Dokumente, generiert daraus einen Strom von Term-Dokument-ID Paaren *(TDocID_stream)* und ruft damit wiederholt *SPIMI-Invert* auf.

```
SPIMI-Invert(TDocID_stream)
 1  output_file ← NewFile()
 2  dictionary ← NewHash()
 3  while (free memory available)
 4  do TDocID ← next(TDocID_stream)
 5     if term(TDocID) ∉ dictionary
 6        then postings_list ← AddToDictionary(dictionary,term(TDocID))
 7        else postings_list ← GetPostingsList(dictionary,term(TDocID))
 8     if full(postings_list)
 9        then postings_list ← DoublePostingsList(dictionary,term(TDocID))
10     AddToPostingsList(postings_list,docID(TDocID))
11  sorted_terms ← SortTerms(dictionary)
12  WriteBlockToDisk(sorted_terms,dictionary,output_file)
13  return output_file
```

Abbildung 2.6: *Pseudocode des Single-Pass In-Memory Indexing Algorithmus*
modifizierte Darstellung von [MRS08, Fig.4.4]

Während eines Durchlaufs von SPIMI-Invert wird jedes *TDocID* Paar separat innerhalb der Schleife behandelt (4ff). Erscheint ein Term erstmalig, so wird er dem Vokabular hinzugefügt und eine neue Posting-Liste erstellt (6). Andernfalls wird in Zeile 7 eine bereits bestehende Liste verwendet. Der reservierte Speicher für diese dynamische Posting-Liste wird nötigenfalls vergrößert (9). Die Liste steht nun zur Verfügung und das aktuell behandelte *TDocID* Paar wird sofort hinzugefügt (10). Wenn der Arbeitsspeicher knapp wird erfolgt eine Sortierung der Terme (11) und der Index für diesen Block wird ausgelagert (12). Die Sortierung vereinfacht die spätere Verschmelzung zum Gesamtindex, die in diesem Teilalgorithmus ebenfalls nicht angeführt ist.

Beliebig große Datenbasis und zugleich effizienter

Im Vergleich zu BSBI ist der durch die Verdoppelung eventuell ungenutzte Speicher für die dynamischen Listen durch den Gewinn aufgrund der entfallenen Struktur zur Term-ID Verwaltung mehr als wett gemacht. Gesamt gesehen steht also ein geringerer Speicherbedarf zu Buche und SPIMI-Invert kann daher wesentlich größere Blöcke verarbeiten. Durch das sequenzielle, unsortierte anhängen neuer Postings an die Liste wird auch ein Geschwindigkeitsgewinn gegenüber dem ersten Verfahren erzielt.

Verteilte Index-Erstellung Internet-Suchmaschinen haben mit dem *World Wide Web* eine enorme Kollektion von Quelldokumenten zu indexieren. Diese Aufgabe lässt sich nicht mehr mit einem einzelnen Computer bewerkstelligen und so kommen häufig ganze Farmen von Standard-Rechnern zum Einsatz. Als logische Konsequenz ist auch der Index auf viele Maschinen verteilt.[1]

MapReduce

Aufgrund dieser Verteilung ergibt sich die Notwendigkeit einzelne Arbeitspakete einfach auf separate Rechner zuweisen zu können. Darüber hinaus müssen solche Zuweisungen für den Fall eines Computer-Ausfalls auch wiederholbar sein. *MapReduce*[2] ist ein Gerüst für verteiltes Rechnen und unterstützt diese Aktionen. Eine typische MapReduce-Anwendung verarbeitet etliche Terabytes an Daten und nutzt dafür etliche Tausende Standard-PCs. [DG08] Abbildung 2.7 veranschaulicht den Einsatz von MapReduce zur Erstellung eines invertierten und verteilten Index.

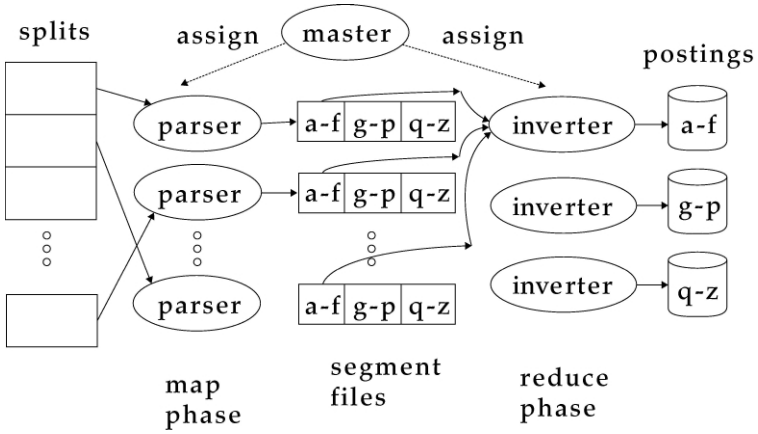

Abbildung 2.7: *Verteilte Index-Erstellung mit MapReduce*
Quelle: [MRS08, Fig. 4.5]

Die Dokumente der Kollektion werden in Gruppen von sinnvoller Größe aufgeteilt, sodass einerseits die Größe einer Gruppe die einfache Verteilung ermöglicht und andererseits die Anzahl der Gruppen handhabbar bleibt. Der *Master* verwaltet und verteilt diese Gruppen. Hat ein Rechner eine Gruppe abgearbeitet, so wird ihm die nächste zugewiesen. Schlägt die Verarbeitung einer Gruppe fehl, so wird sie einem anderen Computer erneut zugewiesen.

Der Name MapReduce bildet sich durch die beiden wesentlichen Phasen.

Map-Phase: Die in Abbildung 2.7 als *Splits* bezeichneten Gruppen werden in Paare aus Term- und Dokument-Identifikator übergeführt. Jeder Parser

1 Abgesehen davon könnte bzw. würde der Index die Kapazitäten eines einzelnen Computers sprengen.

2 http://www.mapreduce.org

generiert dabei für jedes Term-Segment eine eigene Datei. Die Darstellung veranschaulicht das für drei Term-Segmente (a-f), (g-p) und (q-z). Zu jedem Term-Segment gibt es daher gleich viele korrespondierende Dateien wie Parser, also im vorliegenden Beispiel drei Dateien für das Segment (a-f).

Reduce-Phase: Der Master weist jedem Inverter eine eigenes Term-Segment zu. Für jede Term-ID aus diesem Segment fasst der Inverter alle Dokument-IDs zu einer Liste zusammen und sortiert sie nach der Term-ID. So entstehen die sortierten Posting-Listen und damit der invertierte Index.

Weitere Aspekte

Große Datenbasen wie etwa das *World Wide Web* bedingen, dass immense Datenmengen verarbeitet und auch gespeichert werden müssen. Dadurch ergibt sich der Bedarf die Daten — also den Dokumentenspeicher, den Index und auch die ausgelagerten Zwischenergebnisse — zu komprimieren. Damit erschlägt man zwei Fliegen mit einer Klappe:

- Aufgrund der Komprimierung wird weniger Festspeicher und damit weniger Hardware benötigt.
- Geringere Datenmengen beanspruchen weniger Zeit für Schreib-/ Lesezugriffe, reduzieren den internen Datenverkehr der Suchmaschine und bringen somit erhöhte Performanz mit sich.

Rasch veränderliche Dokumentkollektionen verlangen wiederum einen möglichst aktuellen und dynamischen Index. Auch hierfür gibt es einige Ansätze, die jedoch außerhalb des Rahmens dieser Arbeit liegen. Man erkennt, dass das ebenfalls eine ernstzunehmende Herausforderung für Internet-Suchmaschinen darstellt.

Abschnitt 2.4 hat auf den vergangenen Seiten den Bogen von der Text-Akquise über diverse Zwischenschritte und Methoden der Text-Transformation bis hin zum fertigen Index gespannt. Damit ist der Prozess der Indexierung aus Abbildung 2.3 abgeschlossen und der Index bereit für die Suche.

2.5 Suche

Der voranstehende Abschnitt beschreibt den für Benutzer unmerklichen Prozess der Indexierung. Sein Ziel ist die Erstellung des Index als Schnittstelle zum Suchprozess. Die Darstellung 2.2 der Minimalarchitektur auf Seite 17 veranschaulicht diesen Zusammenhang. Die untere Hälfte der Illustration zeigt den Ablauf der Suche. Hier geht es nun darum, die Früchte der zur Indexierung geleisteten Arbeit zu ernten und daraus praktischen Nutzen für den Anwender zu schaffen. In Abweichung zur prozessorientierten Strukturierung bei der Indexierung wird die Beschreibung der Suche nach ihren Komponenten gegliedert. Dadurch wird eine thematische Abgrenzung der Evaluierung ermöglicht.

Bisher wurden Begriffe wie „Suchanfrage“, „Suchbegriff“ und „Query“ meist gleichbedeutend verwendet. An dieser Stelle erscheint nun eine strengere Abgrenzung sinnvoll.

Verknüpfung von Begriffen durch Operatoren

Suchanfrage: Sie bezeichnet jene Zeichenfolge, die der Anwender in der Benutzeroberfläche eingibt. Dabei erlauben viele Anwendungen die Verwendung von *Operatoren*[1] zur genaueren Formulierung der Anfrage. In der Regel lassen sich diese Operatoren durch Zerlegung der Suchanfrage in mehrere Teilanfragen und anschließende Vermengung derer Ergebnisse entsprechend der Operatoren realisieren.[YL96] Die Suchanfrage wird im Weiteren daher als Aneinanderreihung einzelner *Suchbegriffe* betrachtet, zwischen denen ein gedachtes „und/oder“ steht.

Query: Die Summe der *Query-Terme* bildet den Query. Er wird durch Transformation aus der Suchanfrage gewonnen und stellt die Zeichenfolge zum Vergleich mit dem Index dar.

Anders formuliert gibt der Benutzer eine Suchanfrage ein, die aus einem oder mehreren Suchbegriffen besteht. Diese Suchanfrage wird in einen Query transformiert,[2] der wiederum aus einem oder mehreren Query-Termen besteht und von der Suchmaschine verarbeitet werden kann.

Abbildung 2.8 zeigt die wesentlichen Komponenten für den Suchprozesses. Die offenkundig notwendige Benutzer-Interaktion ermöglicht dem Anwender die Eingabe einer Suchanfrage. Der daraus generierte Query wird mit dem Index abgeglichen und die Resultate entsprechend gereiht. Aufbereitung und Präsentation der Ergebnisse sind dann wiederum Bestandteil der Benutzer-Interaktion. Parallel dazu werden die Anfragen zum Zwecke der Evaluierung protokolliert.

1 Verknüpfung von Suchbegriffen mit booleschen Operatoren wie *UND, ODER, NICHT* oder Kennzeichnung von Phrasen mittels Apostrophen...

2 siehe Query-Engine, Abbildung 2.2

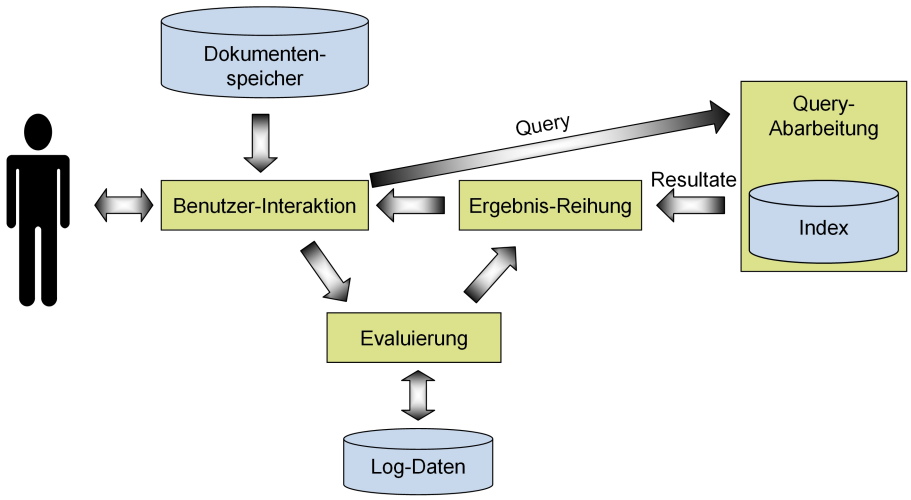

Abbildung 2.8: *Der Suchprozess*
modifizierte Darstellung von [CMS10, Fig.2.2]

2.5.1 Benutzer-Interaktion

Die Benutzer-Interaktion aus Abbildung 2.8 hat die Transformation der Suchanfrage und Darstellung der Ergebnisse zu erfüllen.

Aufbereitung der Suchanfrage

Lässt man implementationsabhängige Optionen[1] außer Betracht, so reduziert sich die Transformation von Suchanfrage zu Query auf die bereits aus Abschnitt 2.4.2 bekannten Methoden. Der dort genannte und begründete Hinweis soll in ähnlicher Form nochmals in Erinnerung gerufen werden:

Es ist Entscheidend, dass bei der Überführung Suchanfrage → Query die selben Methoden Anwendung finden, die auch während der Indexierung zur Transformation Dokument → Index-Terme benutzt werden!

Präsentation der Ergebnisse

Oft ist eine pure Auflistung der Resultate nach absteigender Relevanz schlichtweg unzureichend. Bei der Darstellung der Ergebnisse wird versucht dem An-

1 wie beispielsweise die mögliche Verwendung von logischen Operatoren in der Suchanfrage

wender einen möglichst guten Überblick über die Treffer zu geben. Zu jedem Ergebnis werden daher neben einer Verknüpfung zum Quelldokument meist zusätzliche Informationen dargestellt.

Es sei hier unterstellt dass Suchmaschinen die Ergebnisse in Form von Hyperlinks zur Verfügung stellen. Als Ankertext für die Verknüpfung wird in der Regel der Dokumenttitel verwendet. Häufig werden eine Vorschau und die errechnete Relevanz angeführt. Auch das Dateiformat, die Adresse des Quelldokuments oder die Anzahl der darauf verweisenden Objekte aus der Datenbasis können dargestellt werden. Je nach Anwendungsbereich der Suchmaschine werden dem Anwender noch viele weitere Informationen dargeboten, die von Nutzen erscheinen.

Die Präsentation der Ergebnisse erfolgt ohne Zugriff auf die Quelldokumente! Alle dargebotenen Informationen stammen ausschließlich aus Index und Dokumentenspeicher der Suchmaschine.

Alle zur Ergebnispräsentation herangezogenen Daten haben eines gemeinsam: Sie stammen aus dem Index oder Dokumentenspeicher der Suchmaschine ohne Zuhilfenahme des Basisdokuments! Hier schließt sich der Kreis zur Text-Akquise. Nur Daten die während dieses Prozesses gesammelt wurden stehen zur Darbietung des Resultats zur Verfügung. Der Grund dafür ist leicht erklärt: Der Abruf der Quelldokumente und deren Verarbeitung während der Ergebnispräsentation wären schlichtweg zu langsam. Benutzer erwarten in Sekundenbruchteilen eine Darstellung der n relevantesten Dokumente zu einer Anfrage.

2.5.2 Abarbeitung des Query

Besteht der Query aus einem einzigen Term, so reduziert sich dessen Abarbeitung auf das Auslesen der korrespondierenden Postings aus dem Index und eine allfällige Berechnung der Gewichte.

Ist der Query aus mehreren Termen zusammengesetzt, so gibt es zwei mögliche Ansätze. Nachstehend werden die Ideen hinter diesen Verfahren zum Verständnis gebracht. Den Beschreibungen und Algorithmen liegt *„Search Engines: Information retrieval in practice“* [CMS10] zugrunde.

Dokument-orientiertes Verfahren

Dokument für Dokument

Bei diesem Ansatz wird Dokument für Dokument vorgegangen, wie in Abbildung 2.9 veranschaulicht wird. Diese Methode iteriert also über die Dokumente und summiert in jedem Durchlauf das Gewicht aller Terme für ein Dokument.

```
DocumentAtATime( Query Q, Index I,k)
 1  L ← NewArray()
 2  R ← NewPriorityQueue(k)
 3  for each Term q in Q
 4  do
 5      inverted List l ← GetInvertedList(q, I)
 6      L.Add(l)
 7  for each Document d in I
 8  do
 9      Score s ← 0
10      for each l in L
11      do
12          s ← s + ComputeScore(d, l)
13      R.Add(d, s)
14  return R
```

Abbildung 2.9: *Pseudocode der Dokument-orientierten Query-Abarbeitung*
modifizierte Darstellung von [CMS10, Fig.5.16]

Zunächst wird ein Feld L mit den invertierten Listen der Query-Terme aufgebaut (3-6). Zeile 7 markiert die äußere Iteration über die Dokumente, die Zeilen 9-13 werden also für jedes Dokument durchlaufen. In der Praxis werden verständlicherweise nicht alle indexierten Dokumente untersucht. Stattdessen wird nur jene Teilmenge betrachtet, die in den invertierten Listen l zu den Query-Termen referenziert sind.

maximal k Kandidaten im Speicher

Für ein bestimmtes Dokument wird das Teilgewicht je invertierter Liste — sprich: je Query-Term — errechnet und zum Gesamtgewicht des Dokumentes summiert (12). Dann wird mit dem nächsten Dokument fortgefahren (8) bis alle zu betrachtenden Dokumente abgehandelt sind. Eine Liste R der Dokumente und ihrer Gewichte wird im Speicher verwaltet. Nach jeder äußeren Iteration ist demnach das Gesamtgewicht eines weiteren Dokumentes bekannt. Da im Regelfall nur jene k Dokumente mit höchster Relevanz als Ergebnis returniert werden, ist es ausreichend zu jeder Zeit die k Dokumente mit der bislang höchsten Priorität in R zu verwalten und am Ende diese Struktur zu returnieren.

Geht man davon aus dass aufgrund der Größe von L der Hauptteil der Listen im Festspeicher gehalten wird, so benötigt dieser Algorithmus praktisch nur Hauptspeicher für die Ergebnisliste R. Das ist der wesentliche Vorteil dieses Verfahrens. Nachteilig ist hingegen der Zugriff auf L da hier viele relativ langsame Festplatten-Lesezugriffe auf geografisch nicht benachbarte Daten erforderlich sind.

Term-orientiertes Verfahren

Beim Algorithmus aus Abbildung 2.10 wird Term für Term behandelt. Dieses Verfahren iteriert also über die Terme und bildet in jedem Durchlauf das Gewicht aller dafür gelisteten Dokumente. Dadurch werden die Posting-Listen sequenziell vom peripheren Speicher gelesen und dieser Ansatz eliminiert so die Schwäche der Dokument-orientierten Methode.

Term für Term

```
TermAtATime( Query Q, Index I,k)
 1  A ← NewHashTable()L ← NewArray()
 2  R ← NewPriorityQueue(k)
 3  for each Term q in Q
 4  do
 5      inverted List l ← GetInvertedList(q, I)
 6      L.Add(l)
 7  for each l in L
 8  do
 9      for each Document d in l
10      do
11          A_d ← A_d + ComputeScore(l, d)
12  for each Accumulator A_d in A
13  do
14      R.Add(d, A_d)
15  return R
```

Abbildung 2.10: *Pseudocode der Term-orientierten Query-Abarbeitung*
modifizierte Darstellung von [CMS10, Fig.5.18]

Analog zum Dokument-orientierten Verfahren wird zunächst ein Feld mit allen invertierten Listen erstellt (3-6). Zeile 7 markiert die äußere Schleife, die in diesem Fall über die invertierten Listen iteriert. Die Zeilen 9-11 werden also für jede invertierte Liste l — sprich: für jeden Query-Term — durchlaufen. Für alle in der Liste enthaltenen Dokumente d wird in einem Akkumulator A das Gewicht

sequenzieller Datenzugriff

jedes Dokuments bezüglich dem aktuellen Term errechnet und zu dem Ergebnis aus vorherigen Iterationen summiert (11). Sobald alle Terme und dazu gelisteten Dokumente abgearbeitet wurden, enthält die Hash-Tabelle A für alle gelisteten Dokumente das jeweilige Gewicht. Aus dieser Tabelle muss nun noch eine sortierte Liste der k besten Ergebnisse erstellt und zurückgeliefert werden (12-15).

Dokument- und Term-orientiertes Verfahren unterscheiden sich durch Performanz und Speicherbedarf. Sie führen aber stets zu identischen Ergebnissen!

Der Gewinn dieses Verfahrens liegt in der besseren Performanz durch das sequenzielle Lesen der invertierten Listen. Nachteilig ist der Speicherbedarf für das Feld A der Akkumulatoren.

In der Praxis wird keines dieser Verfahren ohne weitere Optimierung verwendet. Moderne Suchmaschinen arbeiten einen Query meist in mehreren Schritten ab um den Durchsatz zu maximieren und die Antwortzeit zu verkürzen. Dabei wird in der ersten Stufe eine Menge möglicher Ergebniskandidaten aus dem Index bestimmt und nur diese Teilmenge genauer gegen den Query geprüft. Im Kern liegt diesen mehrschichtigen Verfahren meist der Dokument-orientierte Ansatz zugrunde. [YDS09]

2.5.3 Reihung der Ergebnisse

Die Ergebnisreihung ist für Suchmaschinen *der* wesentliche Erfolgsfaktor und zugleich die heikelste Aufgabe. Dabei werden oft mehrere Modelle und Algorithmen miteinander kombiniert um schlussendlich eine stetige Sortierung der Resultate returnieren zu können. [BYRN11] Rund um Internet-Suchmaschinen haben sich mittlerweile ganze Wirtschaftszweige etabliert. Einer der bekanntesten Vertreter ist die *Suchmaschinenoptimierung*. Ihr alleiniger Zweck ist die Beeinflussung der Ergebnisreihung von Suchmaschinen, meist aufgrund wirtschaftlicher Interessen. [ACKC10]

Das verdeutlicht, dass dieser Abschnitt aus Gründen des Umfangs bestenfalls an der Oberfläche dieses Themas kratzen kann. Ein grundlegendes Verständnis dieses Gebietes wird anhand einiger Eckpfeiler vermittelt. Zunächst ist zu entscheiden welche Daten, Modelle und Algorithmen zur Ermittlung der Relevanz herangezogen werden sollen.

Denkbare Daten

Daten, die während der Indexierung gesammelt wurden beziehungsweise bei der Suche daraus ermittelt werden, können in zwei Klassen eingeteilt werden.

Dokument-interne Daten: Diese Klasse umfasst alle Informationen die *im Dokument* enthalten sind beziehungsweise daraus gewonnen werden. Dazu gehören etwa der Titel, Anzahl und Position der Treffer für einen Query-Term oder die Metadaten aus entsprechenden HTML-Tags des Quellobjekts. Auch Gewichtungen, die aus einer Analyse des Dokuments gewonnen werden, gehören zu dieser Sparte.

Dokument-externe Daten: Solche Informationen werden sozusagen *rund um das Dokument* gesammelt. Beispielhaft hierfür sind die Anzahl anderer Datenbasis-Elemente die auf ein Objekt verweisen, sowie die dabei verwendeten Ankertexte. Auch die Klick-Rate[1] fällt unter diese Rubrik.

Wenn ein Algorithmus die Reihung nur anhand Dokument-interner Daten vollführt, ergibt sich eine gewisse Verwundbarkeit. Ersteller von Dokumenten können die Reihung leicht beeinflussen indem sie mit diversen Tricks das Gewicht ihrer Objekte für bestimmte Terme erhöhen. Das ist nicht so einfach möglich wenn die Reihung auf Dokument-externen Daten beruht oder diese zumindest mit einbezieht. [SEW07], [YL96], [Golo8]

Mögliche Modelle

Im modernen Information Retrieval gibt es etliche Ansätze zur Bestimmung der Relevanz. Zwei der bedeutendsten Modelle seinen herausgegriffen und kurz vorgestellt.

Das Vektorraum-Modell betrachtet jeden Term in einem Dokument als eigene Dimension in einem multidimensionalen Vektorraum. Das Dokument ist dann als Vektor $\vec{D}$ in diesem Raum darstellbar. Je öfter ein Term vorkommt, desto größer ist der Betrag des Vektors in dessen Dimension. Analog dazu ist auch ein Query als Vektor $\vec{Q}$ abbildbar. Die Relevanz des Dokuments in Abhängigkeit vom Query ist dann mathematisch durch den Kosinus des Winkels zwischen den Vektoren oder als deren Skalarprodukt berechenbar. Um Dokumente verschiedener Längen besser miteinander vergleichen zu können werden die Vektoren normiert. [MRS08]

Dokument und Query als normierte Vektoren

1 Viele Suchmaschinen protokollieren welche der angebotenen Treffer auch tatsächlich vom Benutzer angeklickt wurden.

Anhand dieses Modells lässt sich neben der Relevanz auch die Ähnlichkeit von Dokumenten untereinander bestimmen. Dadurch können Duplikate aus den Ergebnissen gefiltert werden oder eine Suche nach ähnlichen Objekten ermöglicht werden. Dieses Modell ist daher für Anwendungen mit großen Dokumentkollektionen besonders interessant.

Das Wahrscheinlichkeitstheoretische Modell ermittelt die Relevanz anhand einer Abschätzung wie groß die Wahrscheinlichkeit ist, dass ein Dokument für einen Query relevant ist. Die Reihung erfolgt dann nach absteigenden Wahrscheinlichkeiten. [Sino1] Aus den zuvor genannten Gründen wird das Vektorraum-Modell in dieser Arbeit bevorzugt.

Beispielhafte Algorithmen

Nachfolgende Algorithmen stellen nur einen kleinen Ausschnitt der mannigfaltigen Berechnungsmöglichkeiten dar. Aus didaktischen Gründen werden mathematische Darstellungen gegenüber Pseudocodes bevorzugt. Alle Beispiele verwenden dabei die selbe Notation.

Q	...	Query, bestehend aus M Query-Termen Q_i
D	...	Datenbasis, bestehend aus N Dokumenten D_i
$R(D_i, Q)$	...	Relevanz von D_i für Query Q; Schlüssel für Ergebnisreihung
$C(D_i, Q_j)$	=	1 wenn Query-Term Q_j in Dokument D_i vorkommt, sonst 0
$Li(D_i, D_j)$	=	1 wenn ein *eingehender* Link von D_j zu D_i existiert, sonst 0
$Lo(D_i, D_j)$	=	1 wenn ein *ausgehender* Link von D_i zu D_j existiert, sonst 0

„Boolean Spreading Activation" Algorithmus: Stillschweigend wurde bislang vorausgesetzt dass die Sortierung der Ergebnisse danach erfolgt *wie* relevant ein Dokument für eine Suchanfrage ist. Dadurch wurde die Menge der rationalen Zahlen als Wertebereich für die Relevanz suggeriert.

Basis: Boolesches Retrieval

Dieser Algorithmus geht ursprünglich aus dem Bereich des *„booleschen Retrieval"* hervor, bei dem die Relevanz nur binäre Werte, also 0 oder 1, annehmen kann. Durch eine kleine Erweiterung kann jedoch eine Reihung ermöglicht werden, die darauf basiert *wie viele* der Query-Terme in den einzelnen Dokumenten vorkommen ohne zu bewerten *wie oft* sie auftreten.

$$R(D_i, Q) = \sum_{j=1}^{M} C(D_i, Q_j)$$

Damit ist der kurze Ausflug in die binäre Welt auch schon wieder beendet. Obige Formel verwendet also nur Dokument-interne Daten. *Boolean Spreading Activation* erweitert diesen Ansatz auf referenzierende und referenzierte Dokumente in der Umgebung. Dahinter steckt die Überlegung, dass miteinander verknüpfte Dokumente auch in einer gewissen inhaltlichen Relation zueinander stehen und ein Objekt daher relevant sein kann obwohl es den Query-Term nicht enthält.

$$R(D_i, Q) = \sum_{j=1}^{M} I_{i,j} \quad \text{mit} \quad I_{i,j} \begin{cases} c_1 & \text{wenn ein } k \text{ existiert sodass } C(D_k, D_j) = 1 \\ & \text{und } Li(D_i, D_k) + Lo(D_i, D_k) > 0 \text{ ist} \\ c_2 & \text{wenn } C(D_i, Q_j) = 1 \\ 0 & \text{andernfalls} \end{cases}$$

Eine Gewichtung zwischen untersuchtem, referenzierenden und referenzierten Dokumenten erfolgt anhand zweier Konstanten mit $0 \leq c_1 \leq c_2$. Dieser Algorithmus nutzt demnach die internen Daten des betrachteten Dokumentes sowie externe Informationen — Anzahl und Inhalt der verknüpften Dokumente — für die Berechnung der Relevanz. [YL96]

„Most Cited" Algorithmus: Hier basiert die Reihung gänzlich auf der Relevanz jener Objekte, die auf das betrachtete Dokument referenzieren. Zunächst wird die Relevanz bezüglich des Query für jedes referenzierende Objekt errechnet. Das Gewicht des betrachteten Dokuments wird dann durch die Summe der Gewichte der referenzierenden Objekte ermittelt. [YL96]

$$R(D_i, Q) = \sum_{k=1, k\neq i}^{N} \left(Li(D_i, D_k) \sum_{j=1}^{M} C(D_k, Q_j) \right)$$

Diese Methode verwendet also keine internen Daten des betrachteten Dokuments für dessen Gewichtung. Stattdessen erlangen jene Dokumente höchste Priorität, die von anderen — für den Query relevanten — Objekten am häufigsten referenziert werden.

Term-Frequenz — Inverse Dokument-Frequenz *tf-idf*: Term- und Dokument-Frequenz wurden im Zuge der Indexierung auf Seite 31 bereits beschrieben. In einem ersten Ansatz lässt sich damit die Relevanz wie folgt berechnen:

$$R(D_i, Q) = \sum_{k=1}^{M} \text{tf-idf}(D_i, Q_k) \quad \text{mit} \quad \text{tf-idf}(D_i, Q_k) = \text{tf}_{D_i, Q_k} \cdot \text{idf}_{Q_k}$$

Verwendet man die Betrachtungsweise des Vektorraummodells, so sind Dokument D_i und Query Q als normierte Vektoren $\vec{d}_i$ und $\vec{q}$ darstellbar. Die Relevanz

eines Dokuments berechnet sich dann als Skalarprodukt von Dokument und Query. [MRS08]

$$R(D_i, Q) = \vec{d}_i \circ \vec{q}$$

Gefundene Ergebnisse adäquat zu reihen ist die wichtigste und zugleich schwierigste Aufgabe von Suchmaschinen. Die Bestimmung der dazu verwendeten Daten und Algorithmen ist besonders ausschlaggebend und erfolgskritisch. Oft werden dabei mehrere Methoden miteinander kombiniert und moderne Suchmaschinen versuchen ihre Algorithmen zur Reihung ständig zu verbessern. Dabei wird oft das Verhalten der Anwender protokolliert und mit einbezogen. [BYRN11] Dieser Ansatz stellt einen Bereich der Evaluierung dar.

2.6 Evaluierung

Evaluierung kann im Kontext der Suchmaschinen unter zwei verschiedenen Gesichtspunkten verstanden werden. Einerseits kann *die Suchmaschine* evaluiert werden, andererseits kann eine Bewertung der Suchanfragen und des Anwenderverhaltens *durch die Suchmaschine* erfolgen.

2.6.1 Evaluierung von Suchmaschinen

Henrich nennt in seinem Lehrbuch [Heno8] drei wesentliche Aspekte für die Bewertung von Information Retrieval Systemen im Allgemeinen:

- Funktionsumfang und Benutzerfreundlichkeit
- Laufzeit-, (...) und Speicherplatzeffizienz
- Qualität der Ergebnisse

Eine Bewertung nach den ersten beiden Gesichtspunkten tritt bei praktisch jeder Software-Evaluierung auf. Methoden dazu lassen sich aus anderen Sparten leicht auf Suchmaschinen übertragen. Dabei kommen für Effizienzbetrachtungen beispielsweise die Kennzahlen aus Abschnitt 2.2.1 in Frage.

Der Qualität des Ergebnisses kommt bei Information Retrieval Systemen eine besondere Bedeutung zu. Für eine Aussage darüber scheint an dieser Stelle eine mathematische Darstellung von Präzision und Ausbeute nach Croft & Co

[CMS10] von Seite 15 besser geeignet. A bezeichnet die Menge der abgerufenen und R die Menge der relevanten Dokumente. Der Operator $\cap$ bildet die Schnittmenge zweier Mengen.

$$\text{Präzision} = \frac{A \cap R}{A}$$

$$\text{Ausbeute} = \frac{A \cap R}{R}$$

Ablauf

Nach absteigender Sortierung werden dabei nach jedem *relevanten Ergebnis*[1] Präzision und Ausbeute berechnet. Das wird so lange fortgeführt bis eine gewünschte Ausbeute erreicht wird. Man erhält dadurch für jede Ausbeute einen Präzisionswert. Wiederholt man dieses Vorgehen für mehrere definierte Ausbeuten, so lassen sich die errechneten Punkte als Graph darstellen, in dem auch mehrere Systeme miteinander verglichen werden können.

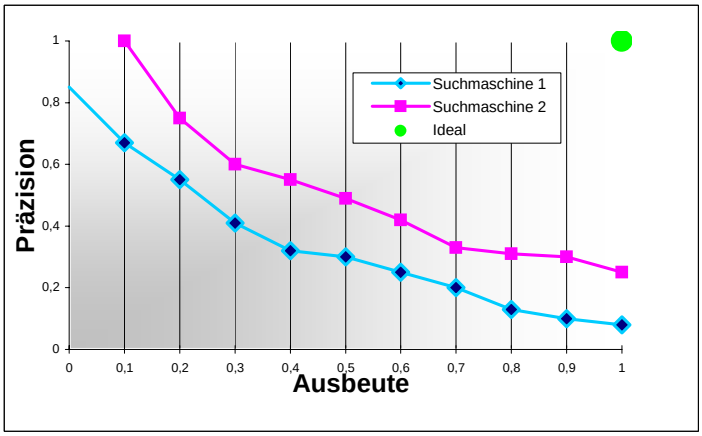

Abbildung 2.11: *Präzision-Ausbeute Diagramm, fiktiver Vergleich zweier Suchmaschinen*

Abbildung 2.11 zeigt ein fiktives Präzision-Ausbeute Diagramm. Der rechte obere Eckpunkt stellt den Idealfall dar, hier sind Ausbeute und Präzision optimal. Suchmaschine 2 würde demnach besser bewertet werden, da die gesamte interpolierte Kurve näher am optimalen Punkt liegt als die von Suchmaschine 1.

Voraussetzung für diesen Vergleich ist, dass die Anzahl der relevanten Dokumente und damit auch die Datenbasis bekannt ist. Für Internet-Suchmaschinen lässt sich die Ausbeute daher nicht berechnen und eine Bewertung der Präzision

1 Die Beurteilung ob ein Ergebnis relevant ist erfolgt meist durch eine Gruppe von Personen.

ist nur anhand der ersten k Ergebnisse möglich. Es ist einleuchtend dass Index-Größe, Indexierungsdauer und andere Maßzahlen nur dann zwischen verschiedenen Implementationen vergleichbar sind, wenn diese die selbe Kollektion indexieren. Das trifft naturgemäß auch für Qualitätsbetrachtungen zu.

Für vergleichende Effizienz- und Qualitätsbetrachtungen von verschieden Suchmaschinen ist es unabdingbar die selben Referenzkollektionen zu verwenden!

Dokument-Kollektionen

In der Praxis haben sich hierfür einige Datenbasen etabliert. Als Pionier ist hier die *Cranfield-Kollektion*[1] zu nennen. Die *GOV2-Kollektion* [2] mit 25 Millionen Websites ist derzeit für wissenschaftliche Zwecke sehr populär. Weitere Kollektionen unter verschiedenen Aspekten — wie beispielsweise Mehrsprachigkeit — haben sich ebenfalls entwickelt. All diese Kollektionen sind jedoch um Potenzen kleiner als die Datenmengen mit denen aktuelle Internet-Suchmaschinen zurande kommen müssen. [MRS08]

2.6.2 Evaluierung durch Suchmaschinen

Die Reihung der Resultate ist eine diffizile und erfolgsbestimmende Angelegenheit. Moderne Suchmaschinen trachten deshalb danach ihr Sortierverfahren stetig zu verbessern. Das Ziel ist also die Aufwertung der Ergebnissortierung.

Eine Möglichkeit ist es vom Benutzer eine explizite Rückmeldung betreffend des präsentierten Resultats einzuholen. In der Regel scheitert dieser Ansatz aber an der Motivation der Anwender. Suchmaschinen stehen jedoch noch alternative Informationsquellen zur Verfügung:

Query-Protokolle lassen sich durch einfache Protokollierung leicht erstellen. Sie vermitteln einen Eindruck über häufig verwendete Suchbegriffe und können somit beispielsweise zur Auto-Vervollständigung bei der Eingabe der Suchanfrage angeboten werden.

Klick-Protokolle lassen sich durch einen „Umweg“ über die Suchmaschine bewerkstelligen. Dabei führen die Verknüpfungen der präsentierten Resultate nicht direkt zum Objekt in der Datenbasis sondern vorerst zurück zur Suchmaschine. Dort wird der Rang des angeklickten Ergebnisses und gegebenenfalls der verwendete Query protokolliert. Der Benutzer wird erst danach zum tatsächlichen Ergebnis weitergeleitet.

1 https://dspace.lib.cranfield.ac.uk/
2 http://www-nlpir.nist.gov/projects/terabyte/

Aus beiden Protokollen kann ein Tripel aus Query, Rang und einer Liste mit der Folge der angeklickten Ergebnisse gewonnen werden. Diese Daten stellen keine absolute Bewertung der Relevanz dar. Jedoch lassen sich daraus Rückschlüsse auf die Reihung ziehen. [Joa02]

Werden für einen bestimmten Suchbegriff beispielsweise überproportional oft die Ergebnisse in der Folge D_1, D_3, D_2 angeklickt, so sollte das auf Platz drei rangierende Dokument eher auf Platz zwei gereiht werden. Basierend auf solchen Überlegungen und Auswertungen lassen sich die Konstanten von Algorithmen anpassen oder Ansätze des maschinellen Lernens implementieren. Derart kann die Ergebnisreihung sukzessive verbessert werden.

2.7 Beispiele moderner Suchmaschinen

In den vorangegangenen Abschnitten wurde auf eine Vielzahl von Hürden hingewiesen, die von modernen Suchmaschinen überwunden werden wollen. Die Quadratur des Kreises zwischen Effizienz, Effektivität und Aufwand, heterogene Quelldokumente, Wahl der Transformation sowie Struktur und Inhalt des Index seien hier nochmals in Erinnerung gerufen.

„Ich weiß, das klingt alles sehr kompliziert... “ Fred Sinowatz † 2008

Letztendlich hängt der Erfolg einer Suchmaschine davon ab wie gut sie diese Aufgaben meistert. Folgender Überblick über einige Implementationen stellt Bezüge zu den vorangegangenen Abschnitten her und rundet damit das Kapitel der modernen Suchmaschinen ab.

2.7.1 Das Lemur-Projekt

Lemur stellt für sich keine dezidierte Suchmaschine dar. Vielmehr handelt es sich um ein im Jahr 2000 gestartetes Open-Source Projekt des „Center for Intelligent Information Retrieval“ der Universität von Massachusetts und des „Language Technologies Institute“ der Carnegie Mellon Universität. Dieses Projekt umfasst folgende Komponenten:

Lemur, das erste Produkt des gleichnamigen Projekts, ist ein Bündel von Werkzeugen und Suchmaschinen zur Unterstützung der Forschung im Umfeld statistischer Modelle für Aufgabengebiete der Informationsrückgewinnung.

Indri ist eine für große textuelle Datenbasen[1] geeignete Suchmaschine die auch einen verteilten Index und Dokumentenspeicher unterstützt. Sie läuft unter verschiedenen Betriebssystemen[2] und indexiert PDF, HTML, XML, und TREC Dokumente. Die Windows-Version verarbeitet auch Word- und PowerPoint Dateien. Die Funktionalität wird über Applikationsschnittstellen in Java, C++, C# und PHP nutzbar gemacht.

Galago unterstützt die textbasierte Suche durch kleine, leicht austausch- und aneinanderreihbare Komponenten sowohl während der Indexierung als auch während der Suche. Mit *TupleFlow* enthält es ein Werkzeug zur verteilten Verarbeitung, ähnlich dem von Seite 35 bekannten MapReduce. Vorhandene Klassen unterstützen invertierte Listen und ermöglichen somit den Aufbau eigener Index-Strukturen ohne dabei von der Pike auf beginnen zu müssen.

Query Log Toolbar ist ein für FireFox und Internet-Explorer verfügbares Plugin um Suchanfragen und besuchte Seiten zu protokollieren. Damit steht ein Werkzeug zur auf Seite 48 beschriebenen Evaluierung *durch* Suchmaschinen zur Verfügung.

ClueWeb09 ist eine große Datenbasis[3] mit über einer Milliarde Dokumenten, die im Jahr 2009 erstellt wurde. Sie kann beispielsweise zur Evaluierung *von* Suchmaschinen (siehe Seite 46) oder zum Vergleich verschiedener Algorithmen und Konfigurationen verwendet werden.

[Lem11]

2.7.2 Google

Heutzutage darf Google[4] wohl in keiner Literatur fehlen, die proprietäre Suchmaschinen auch nur ansatzweise streift. Und so seien auch hier einige Aspekte aufgegriffen, die im Rahmen dieser Masterarbeit erwähnenswert scheinen. Nach Abschnitt 2.1 gehört diese Anwendung zu den proprietären Internet-Suchmaschinen und macht mit seinen Derivaten die Suche nach multimedialen Inhalten möglich.

1 laut eigenen Angaben etwa 500 Millionen Dokumente bei Verwendung mehrerer Server
2 Linux, Solaris, Windows und Mac OSX
3 etwa 5TeraByte im komprimierten, 25TB im unkomprimierten Zustand
4 http://www.google.com/

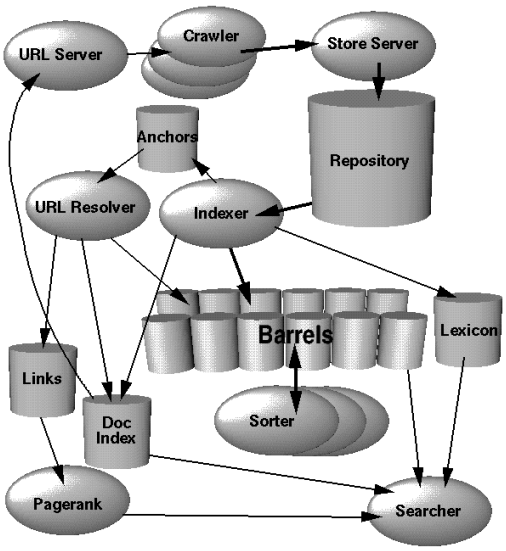

Abbildung 2.12: *Architektur von Google*
Quelle: [PB98]

Google[1] wurde von Larry Page und Sergey Brin ab 1996 entwickelt. Ihr Artikel *„The anatomy of a large-scale hypertextual Web search engine“* [PB98] bietet einen interessanten Einblick in die Entstehungsphase.

Bei der Hardware setzt Google auf weltweit verteilte Farmen von preisgünstigen Standard-Computern und erzielt seine Performanz durch Parallelisierung. Im Jahr 2003 waren dafür bereits über 15.000 PCs im Einsatz. Die Robustheit von Google beruht darauf, dass aus unzuverlässiger Hardware und fehlertoleranter Software eine stabile Infrastruktur geschaffen wird. [BDH03]

Aktuelle Zahlen über die Gesamtheit der verwendeten Hardware werden geheim gehalten aber Spekulationen liegen bei hundert tausenden (!) Computern oder mehr. Diese enorme Parallelisierung schließt auch verteilte Indexe und Dokumentenspeicher mit ein und die Softwarearchitektur muss dementsprechend ausgerichtet sein. So hat Google dem MapReduce[2] zu Popularität verholfen. Weitere Einblicke in die Funktionsweise bietet beispielsweise GoogleGuide.[3]

PageRank und Personalisierung

Wie die meisten Internet-Suchmaschinen verwendet auch Google mit dem eigen-entwickelten *PageRank*-Algorithmus[4] ein *Link-Analyse Verfahren* um die Relevanz eines Dokumentes zu bewerten. [BS09] Bei der Ergebnisreihung wird darüber hinaus der Pertinenz[5] große Bedeutung beigemessen. Zu diesem Zweck wird unter anderem das *Webprotokoll*[6] zur Personalisierung der Suche[7] herangezogen.

Die Popularität von Google sagt wohl einiges darüber aus, wie gut diese Implementation die Anforderungen an moderne Suchmaschinen erfüllt. Andererseits ergeben sich dadurch auch Perspektiven die eine gewisse Kritik rechtfertigen.

1 benannt nach *googol*, dem englischen Begriff für die Zahl 10^{100}
2 siehe Seite 35, http://www.mapreduce.org
3 http://www.googleguide.com
4 benannt nach Larry Page
5 siehe Abschnitt 2.2
6 http://support.google.com/accounts/bin/topic.py?hl=en&topic=14149
7 http://www.google.at/goodtoknow/data-on-google/more-relevant/

Kritik

Eine subjektive Wertung soll hier jedoch nicht das Ziel sein. Vielmehr sei der Leser angehalten sich sein eigenes Bild zu machen und dafür neben den individuellen Erfahrungen und diversen Lobeshymnen auch das Studium kritischer Literatur wie etwa *„Die Google-Falle“* [Rei10] oder *„Deep Search: Politik des Suchens jenseits von Google“* [BS09] in Erwägung zu ziehen.

2.7.3 Wolfram|Alpha

Ausblick in die Zukunft

Was erwartet man sich eigentlich als Ergebnis einer Suche? Diese Frage an diesem Punkt zu stellen mag zwar seltsam anmuten, ist aber durchaus berechtigt. Die intuitive Antwort ist vermutlich, dass die relevantesten Objekte greifbar gemacht werden sollen, die bezüglich einer Suchanfrage *existieren*. Eine alternative Interpretation kann sein, dass man durch eine Suche an relevante Informationen gelangen will. Diese Betrachtungsweise eröffnet die Möglichkeit das Ergebnis zu *generieren*.

Wolfram | Alpha[1] geht als „Computational knowledge engine“ diesen Weg um die Antwort mittels bestehender Informationen und Algorithmen zu generieren. Dieses Langzeit-Projekt folgt Stephen Wolfram's Vision *Jegliches systematische Wissen unmittelbar berechenbar und jedermann zugänglich zu machen*. Dabei wird also versucht die Anfrage zu *verstehen* und das Resultat zu *errechnen*. [Wol12]

Vielleicht müssen wir in Zukunft den Begriff „Suche“ neu überdenken und umfassender verstehen. Zurückkommend auf Suchmaschinen im herkömmlichen Sinn will auch die Open-Source Implementation *Lucene* erwähnt werden. Ihr kommt im Rahmen dieser Arbeit spezielle Bedeutung zu und so sei ihr ein eigenes Kapitel gewidmet.

1 http://www.wolframalpha.com/

3 Lucene

Das vorangegangene Kapitel 2 hat ein allgemeines Verständnis für Suchmaschinen und deren elementare Prozesse der Indexierung und Suche geschaffen. Nun ist es an der zeit ein wenig spezifischer zu werden und anhand eines Beispiels aufzuzeigen wie eine moderne Suchmaschine praktisch umgesetzt werden kann. Analog zur allgemeinen Beschreibung moderner Suchmaschinen erfolgt auch hier eine Unterteilung nach Architektur und den Phasen der Indexierung und Suche.

Die Grundlage für dieses Kapitel bildet das Buch *„Lucene in Action“*. [MHG10] Die drei Autoren *McCandless*, *Hatcher* und *Gospodnetić* verkörpern einen Gutteil des Kern-Entwicklungsteams von Lucene und ihr Werk bietet dementsprechend wertvolle und zuverlässige Informationen aus erster Hand.

Geschichte

Die Wurzeln liegen in Doug Cutting's Vergangenheit bei Xerox PARC,[1] wo er ab 1988 bei mehreren Projekten praktische Erfahrungen im Gebiet des Information Retrieval sammelte. 1997–98 entwickelte Cutting in drei Monaten die erste Implementation von Lucene[2] und legte 2000 den Code bei SourceForge offen. Ein Jahr später wurde das Projekt von der Apache Software Foundation aufgenommen und 2005 zu einem Top-Level Projekt erhoben. Die ursprüngliche Implementation war rein Java-basiert. Mittlerweile ist Lucene auch in anderen Programmiersprachen[3] — wie Perl, Python, Ruby, C, C++ und C# — verfügbar und wird unter anderem bei Wikipedia und in der Eclipse-Entwicklungsumgebung eingesetzt. [Cuto5]

Das *Lucene-Projekt* ist nun ein Ansinnen der Apache Software Foundation zu dem Teilprojekte wie *Nutch*[4], *Solr*[5] oder *Lucy*[6] gehören. Dieses Kapitel legt jedoch den Fokus auf das namensgebende Hauptprojekt *Lucene-Core*, das landläufig als *Lucene* bezeichnet wird.

1 Xerox Palo Alto Research Center, http://www.parc.com/
2 benannt nach dem zweiten Vornamen seiner Frau
3 siehe http://wiki.apache.org/lucene-java/LuceneImplementations
4 erste Open-Source Internet-Suchmaschine, http://nutch.apache.org/
5 gesprochen „Solar“, http://lucene.apache.org/solr/
6 Projekt-Anwärter, http://incubator.apache.org/lucy/

Lucene ist keine vollständige Out-of-the-box Implementation einer Suchmaschine sondern eine *„Open-Source Information Retrieval Library“* mit deren Hilfe textbasierte Suchanwendungen realisiert werden können!

Anwendungsbereich

Lucene beinhaltet die zur Indexierung und Suche notwendigen Kernkomponenten unter deren Verwendung vielfältige Applikationen im Umfeld der textbasierten Suche entwickelt werden können.

Die in Abbildung 3.1 dunkel hinterlegten Bereiche werden dabei von Lucene abgedeckt. Die beinhalteten Klassen und Methoden dienen also dem Prozess von der Indexierung homogener Dokumente bis zur Bereitstellung der Resultate mit allfälligen Formatierungselementen. Darüber hinaus kann es nötig sein die Konfiguration und Steuerung der Kern-Komponenten in der umgebenden Anwendung zu ermöglichen oder die anfallenden Daten zu analysieren. Lucene bietet dazu zwar per se keine eingebetteten Werkzeuge, ermöglicht als Open-Source Projekt aber natürlich deren Anbindung.

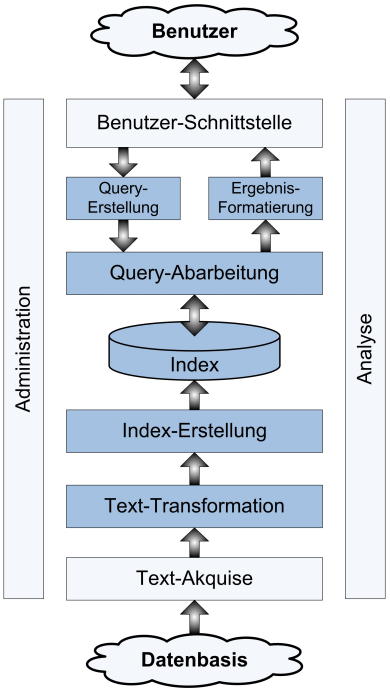

Abbildung 3.1: *Komponenten einer Suchapplikation mit Kennzeichnung der von Lucene abgedeckten Bereiche*
modifizierte Darstellung von [MHG10, Fig.1.4]

Klassifizierung

Durch die performanten Algorithmen und das gute Design ist Lucene in allen Bereichen der textbasierten Suche einsetzbar. Die Bibliothek schränkt die Klassen der damit realisierbaren Applikationen praktisch nicht ein. Die Anwendungsbereiche erstrecken sich folglich von der Desktop-Suche über unternehmensweite Applikationen bis hin zu Internet-Suchmaschinen. Durch gezielte Text-Akquise ist dabei stets auch eine vertikale Implementierung realisierbar.[1]

Die Projekt-Homepage *http://lucene.apache.org/* bietet Download, Dokumentation und weiterführende Links an. Nachstehende Abschnitte erläutern die Architektur und wie Lucene die Indexierung und Suche unterstützt.

1 siehe Abschnitt 2.1 „Klassifizierung“ auf Seite 9

3.1 Architektur

Die Abbildung 3.2 veranschaulicht die Einbindung in eine typische Suchanwendung. Es ist deutlich erkennbar, dass Lucene den technischen Kern der Indexierung und Suche innerhalb der Anwendung kapselt. Dadurch wird die Umsetzung verschiedenartigster Aufgabenstellungen der textbasierten Suche erlaubt.

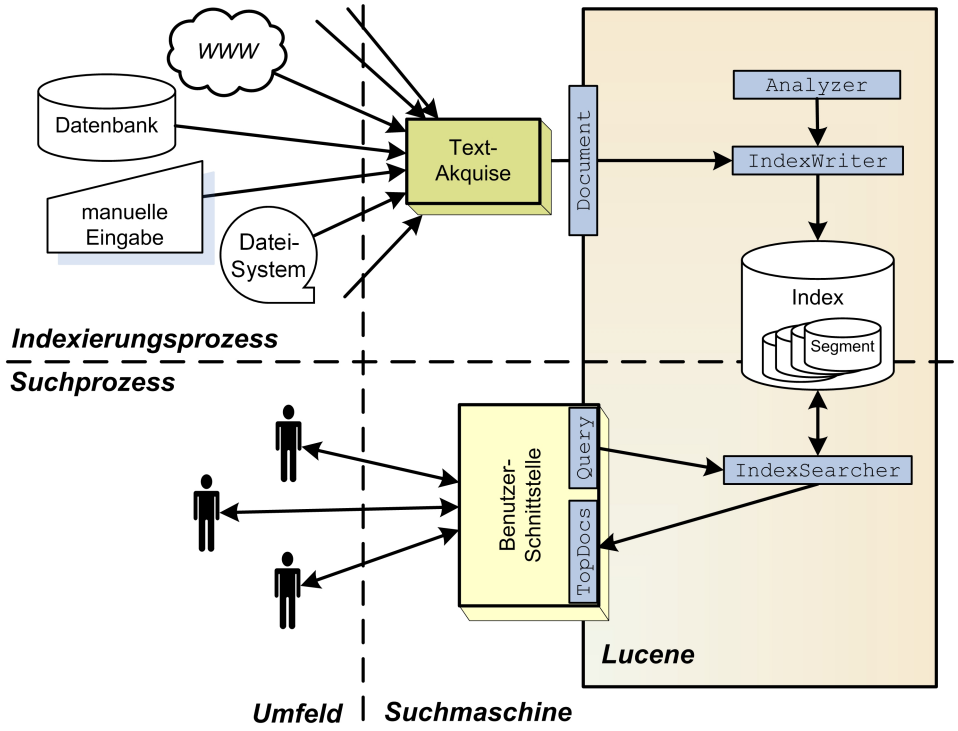

Abbildung 3.2: *Architektur einer Suchapplikation unter Verwendung von Lucene mit Kennzeichnung der fundamentalen Klassen*

Die bläulich hinterlegten Objekte sind fundamentale Klassen der Bibliothek. Sie werden an geeigneten Stellen in den nachfolgenden Abschnitten erläutert.

3.2 Indexierung

Ausgehend von der Datenbasis ist zunächst die Text-Akquise durchzuführen. Sie ist vom konkreten Anwendungsfall abhängig und wird folglich auf Seite 78 behandelt.

Document-Klasse

Die `Document`-Klasse bildet die Schnittstelle zu Lucene. Um Verwechslungen mit den Objekten der Datenbasis zu vermeiden, werden die `Document`-Klasse beziehungsweise deren Instanzen in `Schreibmaschinenschrift` dargestellt. Diese Darstellungsweise wird generell im Fließtext dieser Arbeit für alle (Lucene-) Programmierobjekte beibehalten.

Ein `Document` stellt eine Ansammlung von *Feldern* dar, die ihrerseits den Text, Metadaten, Adresse und andere Informationen des Quelldokuments beinhalten. Technisch gesehen ist ein Feld ein *Name–Wert* Paar, das mit verschiedenen Optionen versehen sein kann. Lucene indexiert die Inhalte dieser Felder entsprechend der spezifizierten Einstellungen. Einige dieser Optionen werden am Ende der Text-Transformation auf Seite 58 gestreift. Beinhaltet ein `Document` mehrere gleichnamige Felder, so werden diese im Index zusammengefügt.

3.2.1 Text-Transformation

Die Bibliothek beinhaltet eine reichhaltige Sammlung von Klassen zur Text-Transformation, die im *Analysis-Package* zusammengefasst sind. Dazu gehören eine abstrakte Basisklasse `Analyzer` sowie diverse zur Transformation hilfreiche Klassen wie `StopFilter`, `LowerCaseFilter` oder `StandardTokenizer` um nur einige zu nennen. [Theo6] Durch Aneinanderreihung ausgewählter Komponenten entsteht eine Klasse, die die gewünschte Text-Transformation durchführt. Individuelle Implementationen können daher einfach realisiert werden. Dankenswerterweise enthält Lucene von Haus aus aber auch implementierte Klassen die zur sofortigen Verwendung geeignet sind. Einige davon seien stellvertretend herausgegriffen.

Whitespace Analyzer

Analyzer-Klassen

Der `WhitespaceAnalyzer` implementiert die einfachste Form der Text-Transformation. Hier werden nur Leerräume wie Zeilenvorschub, Tabulator oder Leerzeichen zur Zerlegung des Textes verwendet. Dabei werden aufeinanderfolgende Leerräume zu einem Trennzeichen gruppiert. Alle so ermittelten Token werden unverändert indexiert. Das bedeutet, dass keine Stoppwörter eliminiert werden und weder Normalisierung noch Kanonisierung stattfinden. Jedes einzelne druckbare Zeichen ist Teil eines Terms oder stellt gegebenenfalls einen eigenständigen Term dar. Tabelle 3.1 zeigt, dass auch der Bindestrich unter Verwendung des `WhitespaceAnalyzer` zu einem Term im Vokabular wird.

Simple Analyzer

Eine Erweiterung der erstgenannten Klasse in zweierlei Hinsicht stellt der `SimpleAnalyzer` dar.

Trennzeichen: Nur Folgen von Buchstaben werden als Token gewertet. Alle anderen Zeichen — wie Leerräume, Ziffern, Satzzeichen und dergleichen — werden als Trennzeichen für die Zerlegung verwendet.

Normalisierung: Die einzelnen Token werden auf einheitliche Kleinschreibung übergeführt.

Im Vergleich zum `WhitespaceAnalyzer` bewirkt der Bindestrich keinen Eintrag im Index. Im Gegenzug ist aber auch die eMail-Adresse in mehrere Terme zerlegt. Man muss sich bewusst sein dass diese Klasse auch Ziffern stillschweigend als Trennzeichen interpretiert und somit Zahlen nicht indexiert werden.

Stop Analyzer

Diese Klasse entfernt zusätzlich zum `SimpleAnalyzer` auch Stoppwörter. Standardmäßig wird hierbei eine in der Bibliothek enthaltene Stoppwortliste verwendet. Es handelt sich dabei um etwa 30 gebräuchliche Wörter in englischer Sprache wie *„a“*, *„to“* oder *„the“*. Bei der Instanziierung kann jedoch auf einfache Art und Weise eine alternative Stoppwortliste angegeben werden. Der `StopAnalyzer` greift dabei seinerseits auf einen Filter — den `StopFilter` — zurück, der die Liste im Konstruktor übernimmt. Instanzen dieser Klasse ermöglichen ebenfalls keine Indexierung von Zahlen.

Standard Analyzer

Der `StandardAnalyzer` stellt unter den vorgefertigten Komponenten zur Text-Transformation die meist verwendete Klasse dar. Er implementiert die Zerlegung in Token aufgrund einer grammatikalischen Analyse.[1] Bei der anschließenden Text-Bereinigung kann eine alternative Stoppwortliste verwendet werden oder für englischsprachige Basisdokumente wiederum die standardmäßige Liste englischer Begriffe gewählt werden. Der `StandardAnalyzer` bietet einen guten Einstieg bei einer erstmaligen Verwendung von Lucene.

1 JFlex, `http://jflex.de/`

Der Vorteil dieser Klasse liegt vor allem darin, dass die folgenden Komponenten erkannt und speziell berücksichtigt werden:

- alphanumerische Zeichenfolgen
- Abkürzungen
- Firmennamen
- Hostnamen von Computern
- Zahlen
- Seriennummern
- Worte mit eingeschlossenem Apostroph

Einige dieser Textbausteine sind dabei für englischsprachige Quelldokumente besser geeignet. Andere — wie eMail-/ IP-Adressen oder alphanumerische Begriffe — funktionieren praktisch in allen Sprachen gleich gut. Auch die Erkennung von Zahlen ist ein großes Plus gegenüber den zuvor genannten Klassen und macht den `StandardAnalyzer` für etliche Anwendungsfälle durchaus attraktiv.

Tabelle 3.1 veranschaulicht die unterschiedlichen Ergebnisse der Transformation eines Beispieltextes unter Verwendung der genannten Klassen. Die Eckigen Klammern dienen der besseren Verdeutlichung der einzelnen Terme. Hier ist ersichtlich, dass der Bindestrich unter Verwendung des `WhitespaceAnalyzer` zu einem eigenständigen Term wird. `SimpleAnalyzer` und `StopAnalyzer` zerlegen den Text auf ähnliche Weise: eMail-Adresse und Firmenname werden zerlegt. Der `StopAnalyzer` eliminiert darüber hinaus das Stoppwort *„the“*. Das augenscheinlich beste Resultat liefert der `StandardAnalyzer`, da er den Bindestrich aussortiert und sowohl Firmenname als auch eMail-Adresse erkennt und als Terme zurück liefert. Die letzten drei `Analyzer` führen die Token auf Kleinschreibung über.

Feld-Optionen

Für die Felder eines `Document` können Optionen bezüglich der Indexierung und Speicherung angegeben werden. Dadurch lässt sich das Verhalten während der Indexierung je Feld spezifisch steuern. So kann ein Feld bewusst an der Text-Transformation vorbei geschleust werden indem der `Field.Index` Enumerator etwa auf *„Not Analyzed“* oder *„Not Analyzed No Norms“* festgelegt wird. In diesem Fall wird der gesamte Feldinhalt als einzelner Token im Index abgelegt und ist als Gesamtes suchbar.

Quelltext:	"The AB&C Corporation - office@abc.com"
`WhitespaceAnalyzer`	[The][AB&C][Corporation][-][office@abc.com]
`SimpleAnalyzer`	[the] [ab] [c] [corporation] [office][abc][com]
`StopAnalyzer`	[ab] [c] [corporation] [office][abc][com]
`StandardAnalyzer`	[ab&c] [corporation] [office@abc.com]

Tabelle 3.1: *Lucene-Analyzer – Vergleich der Index-Terme unter Verwendung von Whitespace-, Simple-, Stop- und StandardAnalyzer*
vergleiche [MHG10, Abschnitt 4.1]

Der Enumerator `Field.Store` zur Steuerung der Speicherung ist ebenfalls nennenswert. Mit ihm kann festgelegt werden ob der Inhalt eines Feldes nur indexiert oder auch gespeichert wird. Dabei werden die Inhalte immer in der ursprünglichen Form — also ohne Transformation — abgelegt. Das ist speziell bei Feldern mit tendenziell kurzem Inhalt wie *Titel* oder *Kurzfassung* interessant, da gespeicherte Felder zur besseren Präsentation der Ergebnisse verwendet werden können.

Unabhängig von der Wahl des `Analyzer` ist das Ergebnis der Text-Transformation eine Menge von Termen, die in `Field`-Instanzen von `Document`-Objekten enthalten sind. Sie werden später das Vokabular des Index bilden.

3.2.2 Index-Erstellung

Üblicherweise speichert Lucene die für das Vektorraummodell nötige Term-Frequenz und die Term-Positionen im Index. Durch Optionen der `Field`-Klasse kann das unterbunden werden und somit bei der Suche ein boolesches Modell zur Anwendung gelangen. Das kann für bestimmte Felder ausreichend sein und die Index-Größe vermindern. [MHG10] Für die folgenden Betrachtungen wird jedoch von einer Verwendung des Vektorraummodells ausgegangen.

Index Writer

Die zentrale Klasse bei der Erstellung des Index ist der `IndexWriter`. Er führt zur Laufzeit die Text-Transformation je Feld in Abhängigkeit von festgelegten

Optionen und spezifiziertem `Analyzer` durch. Weiters verwirklicht er die Inversion von *Dokument* → *Term* auf *Term* → *Dokument* Relationen. Um diese Aufgaben erfüllen zu können implementiert diese Klasse Methoden um `Document`-Objekte hinzuzufügen, zu löschen oder zu aktualisieren.

Wie der Name unschwer vermuten lässt übernimmt der `IndexWriter` auch die Ablage des Index im Festspeicher. Es ist anwendungsspezifisch zu entscheiden welche Felder in einem `Document` enthalten sein können. Jedes `Field` bildet letztendlich einen Bestandteil des Index. Hier sind die diesbezüglichen Überlegungen von Seite 31 zu berücksichtigen.

Gewichtung von Dokumenten und Feldern

Durch die Gewichtung von Dokumenten oder Feldern ist es möglich verschiedene Objekte bei der Suche bevorzugt zu behandeln. Die stärkere Gewichtung von Objekten erlaubt indirekt die Einflussnahme auf die Ergebnisreihung. Lucene ermöglicht sowohl eine Dokument- als auch Feld-spezifische Gewichtung.

Die einzelnen Objekte können dabei bereits bei der Indexierung verschieden stark bewertet werden. Auf diese Weise ist es möglich ausgewählten Dokumenten den Vorzug zu geben. Das kann beispielsweise aufgrund der Aktualität eines Dokuments oder dessen Quelle gewünscht sein. Außerdem kann auch eine Gewichtung dynamisch bei jeder Suchanfrage vorgenommen werden. Somit kann dem Benutzer die Möglichkeit geboten werden, seinen Bedürfnissen entsprechend steuernd in die Ergebnisreihung einzugreifen. [Kono8]

Die Gewichtung während der Suche erlaubt mehr Flexibilität, da sie bei jeder Anfrage adaptiert werden kann. Eine Bewertung während der Indexierung minimiert hingegen den Rechenaufwand bei der Suche. Einerlei welche Methode verwendet wird, übertriebenes Eingreifen in die Rangfolge führt tendenziell zu minderer Zufriedenheit der Benutzer! Je stärker die Ergebnisreihung beeinflusst wird, umso schwieriger ist sie für den Anwender nachzuvollziehen und um so weniger intuitiv erscheint das Resultat.

Segmentierung

Jeder durch Lucene erstellte Index besteht aus einem oder mehreren *Segmenten*. Jedes Segment repräsentiert dabei einen eigenständigen Index für eine Teilmenge der Quelldokumente. Bei jedem Schreibzugriff des `IndexWriter`, bei dem die Puffer aus dem Hauptspeicher auf den peripheren Speicher übertragen werden, wird ein Segment erzeugt, das wiederum aus mehreren Dateien besteht.

Dabei beinhaltet eine Datei die Term-Vektoren, eine andere Datei die gespeicherten Felder und so weiter. Alternativ kann — unter geringen Einbußen betreffend der Performanz — festgelegt werden, dass mehrere Bestandteile in einer konsolidierten Datei zusammengefasst werden. So wird während der Suche die Anzahl der simultan benötigten Datei-Deskriptoren minimiert.

Eine bestimmte Datei enthält Referenzen zu allen gültigen Segmenten und wird bei der Suche zuerst konsultiert. Des Weiteren führt der `IndexWriter` bei inkrementeller Indexierung periodisch eine Verdichtung der Segmente gemäß einer *MergePolicy* durch. Hierbei werden Segmente durch den `MergeScheduler` zusammengeführt, obsolete Einträge gelöscht und überzählige Dateien entfernt. Dadurch wird die Performanz gesteigert und die Index-Größe reduziert.

Verteilter Index

Mit Lucene lassen sich zwei Methoden zur Verteilung des Index verwirklichen. [Kono8]

Teilung anhand der Datenbasis: Dabei wird auf mehreren Rechnern jeweils ein Index für eine Teilmenge der Dokument-Kollektion erstellt. Suchanfragen werden dann auf jeden einzelnen Index angewandt und die Resultate zusammengefasst. Die Index-Erstellung kann bei dieser Methodik simultan erfolgen und eine Zusammenführung der einzelnen Indexe wird nicht vorgenommen.

Teilung anhand der Terme: Hier wird der Index alphabetisch nach Termen geteilt. Es entstehen dadurch mehrere Indexe, von denen jeder einen alphabetischen Bereich abdeckt. Die Suchanfrage wird gemäß einer gewarteten Tabelle auf jenem Teilindex abgearbeitet, dessen alphabetischer Term-Intervall den Query beinhaltet.

Bei Internet-Suchmaschinen sind besser skalierende Verteilungen gefragt, die auch Ausfallsicherheit mit sich bringen. Das kann durch Verwendung spezieller verteilter Dateisysteme wie beispielsweise *„Hadoop Distributed File System HDFS“*[1] sowie zusätzlicher Mechanismen für simultanen Schreibzugriff gelöst werden. Lucene erleichtert solche Ansätze indem bei der inkrementellen Indexierung Felder mit den Änderungen an den Index anfügt werden. Das verteilte Dateisystem muss daher statt dem gesamten Teilindex nur dessen Änderungen replizieren. [BRo8]

Ergänzend sei noch erwähnt, dass Lucene eigene Methoden beinhaltet um Zahlen, Daten und Zeiten gesondert zu indexieren. Auf diese Art können spezielle

1 eingesetzt bei Nutch, `nutch.apache.org`

Suchfunktionen — wie eine Produktsuche unter Angabe eines von/bis-Preises — realisiert werden.

Kompression

Mit der Größe der Datenbasis steigt naturgemäß das Verlangen die Index-Größe durch Kompression möglichst klein zu halten. Lucene verfolgt hierfür mehrere Ansätze.

Datentyp von variabler Länge: Ein Großteil der Daten, wie Term-Frequenz, Identifikatoren und viele mehr, sind natürliche Zahlen. Eine effiziente Speicherung dieser Daten wird durch die Verwendung des Datentyps *VInt* ermöglicht. Dabei werden die natürlichen Zahlen nicht mit einer festen Byte-Zahl sondern mit variabler Länge gespeichert. Dabei gibt jeweils das höchstwertige Bit an, ob ein weiteres Byte zur Zahl gehört. Dadurch werden Werte von 0–127 in einem Byte, 128–16383 in zwei Bytes [...] kodiert. Auf diese Weise wird der Speicherplatzbedarf verringert ohne einen aufwändigen Algorithmus zur Kompression beziehungsweise Dekodierung zu benötigen. [Theo6]

Kompression gespeicherter Felder: Lucene bietet die `CompressionTools`-Klasse zur Kompression an. Deren Methoden verwenden den *Deflate* Algorithmus zur Kompression des Feldinhaltes, der beispielsweise auch im ZIP-Format verwendet wird. Die Kompression beim Speichern und Dekompression beim Auslesen der Feldinhalte müssen explizit im Code ausgeführt werden. Der wesentliche Nachteil bei der Kompression von Feldern ist, dass ihre Inhalte nicht indexiert werden und daher nicht durchsuchbar sind.

Kompression des Index: Ein alternativer Lösungsansatz zur Minimierung des Speicherplatzbedarfs ist es, statt dem Inhalt einzelner Felder den gesamten Index im Dateisystem zu komprimieren. *LuceneTransform*[1] ist ein Projekt das diesen Weg verfolgt. Alle Zugriffe auf den Index werden dabei über eine eigene `Directory`-Klasse realisiert. Nach eigenen Angaben lassen sich damit Kompressionsraten zwischen 50 und 90 Prozent erzielen. [Len11]

1 http://code.google.com/p/lucenetransform/

3.3 Luke

An dieser Stelle scheint es passend ein Werkzeug vorzustellen mit dem man sozusagen *in den Index* schauen kann. Luke, die *„Lucene Index Toolbox“* macht von Lucene erstellte Indexe in einer eigenständigen Java-Applikation zugänglich. Unter anderem werden dabei folgende Aktivitäten unterstützt: [Bia11]

- Durchstöbern und Löschen von `Document`-Objekten
- Abruf der Terme mit höchster Frequenz
- ausführen einer Suche sowie analysieren und durchforsten der Ergebnisse
- Zugriff auf verteilte Indexe im Hadoop Dateisystem

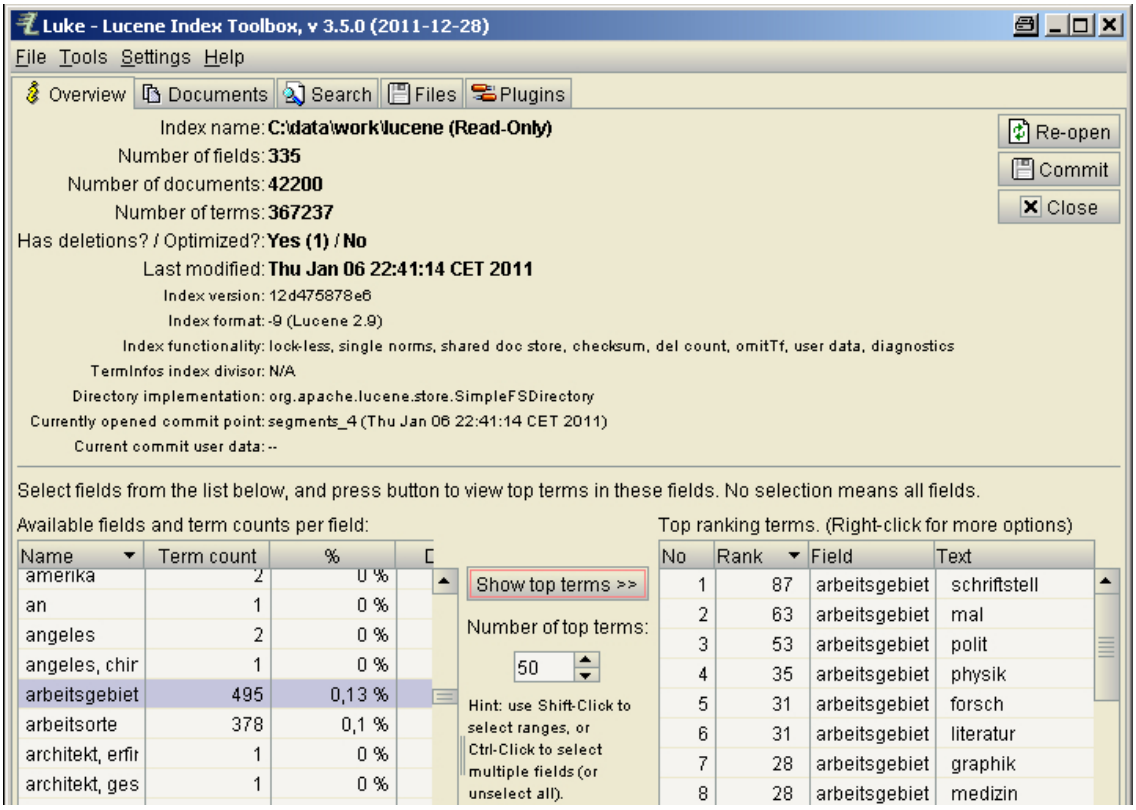

Abbildung 3.3: *LUKE - Benutzeroberfläche der Toolbox, die Lucene-Indexe in einer eigenständigen Anwendung zugänglich macht*

Entwickelt wurde die Applikation von Andrzej Bialecki und mittlerweile gehört sie zum „Lucene development toolkit“. Luke stellt eine enorme Hilfe bei der Entwicklung von Suchmaschinen unter Verwendung der Lucene-Bibliothek dar.

Der Quellcode ist ebenfalls frei verfügbar.[1] Abbildung 3.3 zeigt, dass mit dieser Anwendung der Index sprichwörtlich ein Gesicht bekommt. Dieses Werkzeug visualisiert also das Ergebnis der Indexierung und bietet Hilfestellung bei der Entwicklung und Fehleranalyse einer Suchanwendung.

3.4 Suche

Auch für die Suche sind die benötigten Klassen im Umfang der Lucene-Bibliothek enthalten. Die Gliederung erfolgt zur direkten Vergleichbarkeit analog zum Abschnitt 2.5 im Kapitel der modernen Suchmaschinen.

3.4.1 Benutzer-Interaktion

Wie bereits erwähnt und in Abbildung 3.1 veranschaulicht, ist die Benutzeroberfläche Teil der umgebenden Anwendung. Die Interaktion mit Lucene beschränkt sich demnach auf die Schnittstellen zur Aufbereitung der Suchanfrage und Bereitstellung der Resultate.

Aufbereitung der Suchanfrage

Query-Klasse

Die vom Anwender eingegebene Suchanfrage muss in ein Lucene-taugliches Objekt umgewandelt werden. Die Bibliothek bietet dafür einen `QueryParser` an, bei dessen Instanziierung das Standard-Feld für die Suche anzugeben ist. Mit seiner Hilfe wird die Suchanfrage in eine `Query`-Instanz übersetzt. Für die Überführung der Suchbegriffe in Query-Terme benötigt der `QueryParser` wiederum einen `Analyzer`. Zweckmäßigerweise sollte er mit der bei der Indexierung verwendeten Klasse übereinstimmen.[2]

Lucene bietet einen Schwarm verschiedener Query-Klassen wie `BooleanQuery`, `TermQuery`, `PhraseQuery` oder `NumericRangeQuery`, um nur einige zu nennen, an. Der entsprechende Typ oder eine Kombination mehrerer Typen wird vom `QueryParser` automatisch bestimmt. Dabei werden `Term`-Objekte erstellt und kombiniert. Diese Objekte sind *Name-Wert*-Paare und der `Field`-Klasse sehr ähnlich. Abhängig von der Suchanfrage können sehr komplexe Query-Gebilde entstehen.

1 http://code.google.com/p/luke/
2 Ausnahmen bei Verwendung hoch spezifischer Text-Transformationen bestätigen die Regel.

Eine anderer Ansatz ist es, den `Query` programmatisch zu erstellen. Diese Methode kann für spezielle Aufgaben innerhalb der Applikation von Vorteil sein. Bei der programmatischen Erstellung ist darauf zu achten, dass die Query-Terme die selben Transformationen erfahren wie der Quelltext bei der Indexierung. Für diese direkte Instanziierung stehen auch `Query`-Klassen zur Verfügung, die vom `QueryParser` nicht unterstützt werden. Somit lassen sich spezielle Anforderungen realisieren, für welche die automatische Transformation nicht mehr ausreichend ist.

Gewichtung

Ähnlich wie `Document` und `Field` bietet auch das `Query`-Objekt die Möglichkeit einzelne Terme im Kontext der Anfrage zu gewichten und somit bei der Reihung der Resultate gegenüber anderen Teilen des Query zu bevorzugen. Diese Gewichtung erfolgt bei der Suche und kann praktisch bei jeder Anfrage dynamisch adaptiert werden.

Die Abarbeitung des `Query` geschieht im Kern von Lucene und wird im Abschnitt 3.4.2 gesondert beschrieben. Aus Sicht der Benutzer-Interaktion kann der Ablauf zwischen Query-Erstellung und Erhalt der Resultate vorerst als Mysterium betrachtet werden, das den Query entgegen nimmt und entsprechende Resultate returniert. Die Benutzer-Schnittstelle bietet dem Anwender die erhaltenen Ergebnisse schlussendlich dar.

Präsentation der Ergebnisse

Auf Seite 58 wurde erwähnt, dass mit Hilfe der Feld-Optionen festgelegt werden kann wie Felder bei der Indexierung gehandhabt werden. Dabei gibt es auch die Möglichkeit den Inhalt eines Feldes zwar zu indexieren aber nicht zu speichern. Naturgemäß kann bei der Darstellung der Resultate nur auf Inhalte zurückgegriffen werden, die auch gesichert wurden. Das ist zu beachten wenn beispielsweise eine Vorschau der Resultate angezeigt werden soll.

Zur der Präsentation der Ergebnisse stehen nur jene Felder zur Verfügung, die *indexiert und auch gespeichert* wurden!

TopDocs-Klasse

Die Suchergebnisse werden von Lucene als `TopDocs`-Objekt bereitgestellt. Aus Ressourcen- und Performanz-Gründen handelt es sich dabei verständlicherweise nicht um eine Ansammlung von `Document`-Objekten, sondern um einen Container der Dokument-Identifikatoren als Referenzen zu den Treffern beinhaltet. Diese Referenzen sind nach absteigender Relevanz sortiert. Anhand der Dokument-ID kann auf die einzelnen Treffer zugegriffen werden.

mehrere Ergebnisseiten

Es kommt häufig vor, dass nur m der N Resultate gleichzeitig auf der Ergebnisseite dargestellt werden. Dadurch ergibt sich die Notwendigkeit in mehreren Ergebnisseiten blättern zu können. Prinzipiell gibt es hierfür zwei Konzepte zur Umsetzung:

- das gesamte `TopDocs`-Objekt am Leben erhalten
- den Query erneut abarbeiten

Der erste Ansatz kann zu Ressourcen-Problemen führen wenn viele Benutzeranfragen simultan auf einem Server abgehandelt werden. Die zweite Alternative wird daher oft bevorzugt, zumal Lucene eine Methode bietet um die nächsten m Resultate unter einer gewissen Relevanz-Schranke[1] abzufragen.

Die `TopDocs`-Klasse bietet neben einer Liste mit den Identifikatoren und Gewichten der einzelnen Ergebnisse auch die Gesamtanzahl der Ergebnisse und die maximale Relevanz unmittelbar an. Diese Daten können bei der Darstellung der Ergebnisse ebenfalls mit einbezogen werden.

3.4.2 Abarbeitung des Query

IndexSearcher-Klasse

Die `IndexSearcher`-Klasse stellt das Ambivalent zum `IndexWriter` dar und greift nur lesend auf den Index zu. Dabei werden sowohl gleichzeitige Zugriffe verschiedener Instanzen als auch multiple Threads erlaubt. Unter Angabe des Pfades zum Index wird eine Instanz erzeugt, die dann verschiedene Methoden zur Durchführung der Suche bereitstellt. Im einfachsten Fall wird ein `Query` und die Anzahl der maximal zu returnierenden Ergebnisse übergeben. Der `IndexSearcher` evaluiert den `Query` gegen jedes Index-Segment und kombiniert anschließend die Ergebnisse.

Der tatsächliche Zugriff auf den Index erfolgt mittels eines `IndexReader`. Seine Instanziierung ist aufwendig und so verwenden `IndexSearcher` meist eine bereits bestehende Instanz davon. Dabei ist zu beachten, dass der `IndexReader` stets in einem Abbild sucht, das den Stand des Index zum Zeitpunkt der Öffnung repräsentiert. Erfolgen Indexierung und Suche zeitgleich, so muss der `IndexReader` erneut geöffnet oder instanziiert werden um auch aktualisierte Dokumente im Index zu finden.

Der `IndexSearcher` stellt mehrere Funktionen zur Suche bereit. Neben der zuvor thematisierten Methode gibt es beispielsweise Überschreibungen mit optionalen `Filter`- oder `Sort`-Objekten, die es erlauben nur eine bestimmte Teilmenge der

1 beispielsweise die Relevanz des letztgereihten Ergebnisses auf der aktuellen Ergebnisseite beim Vorwärts-Blättern

potenziellen Ergebnisse abzufragen oder diese alternativ zu reihen. Die sortierten Resultate werden in Form des auf Seite 65 vorgestellten `TopDocs`-Objektes zur Verfügung gestellt.

3.4.3 Reihung der Ergebnisse

Lucene kombiniert zwei Modelle um die Relevanz eines Dokumentes zu errechnen. Mit Hilfe des *booleschen Modells* wird eine Vorauswahl der infrage kommenden Dokumente anhand der Query-Terme getroffen. Für diese Kandidaten wird mittels *Vektorraummodell* die jeweilige Relevanz für die Sortierung der Resultate errechnet. Hierbei kommt das *tf-idf*-Modell von Seite 45 zum Einsatz. Die dort verwendete Notation wird sinngemäß beibehalten. Die Vektorschreibweise verdeutlicht die Bildung des Skalarprodukts.

Q	...	Query bestehend aus M Termen (Dimensionen) Q_k
D_i	...	jenes der N Dokumente für das die Relevanz berechnet wird
$R(D_i, Q)$	...	Relevanz des Dokuments D_i für Query Q
$\cdot$	...	Multiplikation
$\circ$	...	Skalarprodukt

Die Grundlage bildet das Skalarprodukt aus $\vec{D}_i$ und $\vec{Q}$.

$$R(D_i, Q) = \vec{D}_i \circ \vec{Q}$$

Eine euklidische Normierung von $\vec{D}_i$ würde jegliche Information über die Länge des Dokumentes entfernen, was in der Praxis als problematisch erscheint. Eine alternative *Längennorm(D_i)* wird eingeführt um verschiedene Dokumentlängen dennoch rechnerisch berücksichtigen zu können. Die Dokument-Gewichtung aus der Indexierung wird durch den Faktor G_{D_i} berücksichtigt. Der Query wird euklidisch auf $\vec{q}$ normiert und die Gewichtung der Query-Terme bei der Suche ist durch G_Q abgebildet. Eine Korrelationsfunktion $k(D_i, Q)$ integriert die Information wie viele der Query-Terme im Dokument enthalten sind.

konzeptionelle Formel

Mit diesen Erweiterungen ergibt sich die konzeptionelle Formel für die Relevanz wie folgt:

$$R(D_i, Q) = k(D_i, Q) \cdot G_Q \cdot \vec{q} \circ \vec{D}_i \cdot \text{Längennorm}(D_i) \cdot G_{D_i}$$

praktische Formel

Überlegungen bezüglich Performanz und die Berücksichtigung der Feld-Gewichtungen bei der Indexierung führen zur praktischen Formel für die Relevanz:

$$R(D_i, Q) = k(D_i, Q) \cdot \text{QueryNorm}(Q) \cdot \sum_{k=1}^{M} \text{tf}_{D_i,Q_k} \cdot \text{idf}_{Q_k}^2 \cdot G_{Q_k} \cdot \text{Norm}(D, Q_k)$$

Die einzelnen Bestandteile sind dabei:

Korrelationsfunktion $k(D_i, Q)$: Besteht der Query beispielsweise aus drei Termen, so ergibt die Korrelationsfunktion den Wert $\frac{2}{3}$ für ein Dokument das zwei dieser drei Terme beinhaltet.

QueryNorm(Q): Sie ermöglicht einen besseren Vergleich der Ergebnisse für verschiedene Queries und berechnet sich als $\frac{1}{\sqrt{\sum_{k=1}^{M} G_{Q_k}^2}}$. Die Summe der quadratischen Gewichte wird durch das `Query.Weight`-Objekt gebildet.

Term-Frequenz tf_{D_i,Q_k}: Lucene verwendet die Quadratwurzel als Dämpfungsfunktion für die Anzahl der Worterscheinungen in einem Dokument.

Inverse Dokument-Frequenz idf_{Q_k}: Sie gewichtet jene Terme stärker, die seltener in Dokumenten enthalten sind. Bei N Dokumenten im Index, von denen n den Term Q_k enthalten, berechnet sie sich als $1 + \log\left(\frac{N}{n+1}\right)$ Die Additionen von 1 verhindern dabei eine Division durch Null und dass die inverse Dokument-Frequenz für $n = N$ zu Null wird.

Gewichtung G_{Q_k}: Einflussnahme der Gewichtung der einzelnen Query-Terme bei der Suche.

Normierung Norm(D, Q_k): In der Norm werden Dokument- und Feld-Gewichtung aus der Indexierung sowie die ebenfalls während der Indexierung berechnete Längennorm des Dokuments zusammengefasst.

[Kono8], [Theo6]

Explanation Klasse

Die `IndexSearcher`-Klasse von Lucene implementiert die `explain`-Methode. Unter Angabe eines Dokument-Identifikators und eines `Query` erstellt diese Funktion eine `Explanation`. Dieses Objekt legt die Berechnung der Relevanz offen und bietet damit eine komfortable Möglichkeit um die Berechnung nachvollziehen zu können.

4 Das Austria-Forum

Die vorangegangenen Kapitel haben ein grundlegendes Verständnis für den komplexen Themenbereich moderner Suchmaschinen geschaffen und einen Einblick in die Open Source Bibliothek Lucene gewährt, die die Umsetzung von Suchfunktionalitäten in einer umgebenden Anwendung ermöglicht. Das Austria-Forum stellt für die folgenden Kapitel dieser Masterarbeit den praktischen Anwendungsfall dar.

4.1 Austria-Forum - was ist das?

AEIOU

Österreich wurde im Jahr 996 zum ersten Mal namentlich als *Ostarrîchi* in einer Schenkungsurkunde von Kaiser Otto III. für Bischof Gottschalk von Freising erwähnt. Zum tausendjährigen Jubiläum wurden alle beim damaligen Bundesministerium für Wissenschaft und Forschung eingereichten Projekte zum Thema „Kultur-Daten in elektronisch abrufbarer Form“ zu einem Projekt zusammengefasst. In diesem Rahmen wurde von Wissenschaftlern und Experten das *Annotierbare Elektronische Interaktive Oesterreichische Universal-Informationssystem AEIOU*[1] entwickelt. [AEI]

In diesem öffentlich zugänglichen System waren Informationen mit Österreich-Bezug in einem Lexikon und diversen *Alben* — wie Musik-, Video- oder Designer-Album um nur einige zu nennen — organisiert.

Die inhaltliche Basis für AEIOU bildete das 1995 veröffentlichte, zweibändige Österreichlexikon. Aufgrund des schwindenden Interesses und der geringer werdenden finanziellen Unterstützung durch das Ministerium musste im Jahr 2006 eine Entscheidung gefällt werden: AEIOU veralten lassen oder einen umfassenden, technologischen und inhaltlichen Neustart versuchen? Die Entscheidung fiel zugunsten eines Neustarts und im Oktober 2009 konnte das Austria-Forum eröffnet werden. [MM12]

1 http://www.aeiou.at

Das Austria-Forum[1] ist also das Nachfolgesystem von AEIOU und kann nach Trattner [Tra09] als *zitierfähige, Community-geprägte Online-Enzyklopädie mit vorwiegend österreichischen Inhalten* charakterisiert werden. Hinter der Idee steht mit dem *Verein der Freunde des Austria-Forums*, allen voran Em.Univ.-Prof. Dr.phil. Hermann Maurer, eine Gruppe von ehrenamtlichen Mitgliedern. Es besteht eine enge Kooperation mit der Technischen Universität Graz sowie zahlreichen Partnern aus Wissenschaft und Wirtschaft. [AFI10],[Aus] Darüber hinaus kann sich jede interessierte Person kostenlos als Autor registrieren und zum Forum beitragen.

Ziel

> *„Das Austria-Forum ist eine Sammlung von Beiträgen zu allen wichtigen „Austriaca", d.h. von allen Dingen, Personen und Ereignissen, die einen Zusammenhang mit Österreich oder Österreichern haben. Die Beiträge sind qualitätsgeprüft und auf Dauer angelegt, d.h. tagesaktuelle Ereignisse werden bewusst ausgeklammert. Qualitätsgeprüfte Beiträge sind zitierbar wie gedruckte Publikationen: sie haben eine identifizierbare Quelle und ändern sich zeitlich nicht."* [MD10]

Im Wesentlichen gliedert sich das Austria-Forum in vier Bereiche:

AEIOU Österreich Lexikon, basierend auf dem überarbeiteten Datenbestand des AEIOU, strebt eine hohe Qualität und Zitierfähigkeit durch redaktionelle Prüfung und Sperre der Beiträge an.

Redaktionell erstellte Wissenssammlungen, die Speziallexika, enthalten vom Austria-Forum Team erarbeitete und gewartete Beiträge. Diese sind zumeist qualitätsgeprüft und nach einer allfälligen Sperre ebenfalls zitierbar.

Der Communitybereich beinhaltet ausschließlich von Benutzern[2] erstellte Beiträge, die ebenfalls in diverse Wissenssammlungen unterteilt sind. Jeder Besucher kann sich registrieren und in diesem Bereich Artikel bearbeiten oder neue Einträge erstellen.

Allgemeiner Bereich: Informationen über das Austria-Forum an sich, Statistiken, Benutzerhinweise und weitere systembezogene Informationen finden sich in diesem Bereich.

Alle Beiträge im System können durch Benutzer kommentiert werden. [Aus]

1 http://austria-forum.org
2 Anwender die nicht zur Editoren-Gruppe des Austria-Forums gehören

Abbildung 4.1: *Startseite des Austria-Forum*

Für jedermann frei zugängliche und inhaltlich mitgestaltbare Enzyklopädien wie Wikipedia[1] skalieren naturgemäß sehr gut. Mit dem Web 2.0 hat sich die Rolle des durchschnittlichen Internet-Benutzers vom reinen Konsumenten unter anderem durch aktive Teilnahme an solchen Systemen auch zum Produzenten verschoben. Die Qualität der „Veröffentlichungen“ variiert dabei jedoch stark. Die Richtigkeit der Beiträge basiert auf der „Weisheit der Masse“ — je öfter ein Artikel von verschiedenen Benutzern editiert wird, umso geringer ist die Wahrscheinlichkeit von inhaltlichen Fehlern. Bemühungen um die Inhalte solcher Systeme redaktionell zu verifizieren scheitern an der immensen Datenmenge und der ihr gegenüberstehenden vergleichbar geringen Anzahl von Experten und deren Motivation. Im Austria-Forum wird eine Kombination aus den Vorteilen der riesigen Benutzergemeinschaft und der Einbeziehung von Experten zur Qualitätssicherung angestrebt. Dies ist aufgrund des abgegrenzten Themenbereichs — inhaltlicher Bezug zu Österreich und Verzicht auf tagesaktuelle Themen — möglich. Die Qualität der Beiträge steht dabei gegenüber der Quantität im Vordergrund. [HMW08], [MD10]

1 http://www.wikipedia.org

Inhalt und Umfang

Im Austria-Forum wurde der Datenbestand des AEIOU um mehr als 30 Wissenssammlungen erweitert und der Umfang wächst stetig weiter an. Folgende Eckdaten vermitteln einen Eindruck über den aktuellen Umfang dieser Plattform:

- ein allgemeines Lexikon mit etwa 34.000 Einträgen
- ein Musiklexikon mit Noten und Tonbeispielen
- ein Volkskundelexikon mit rund 870 Artikeln
- 1400 Biographien
- 680 Videos aus verschiedensten Themenbereichen
- 1400 historische Bilder und 3600 Fotos aus allen Bundesländern
- 2000 Einträge zu heimischen Tieren und 1500 Einträge zu beheimateten Pflanzen
- alle Briefmarken der zweiten Republik
- digitale Bücher zum Blättern und Online-Lesen
- 4000 Einträge zu Bildbänden und Kunstbüchern
- und vieles mehr...

[Die11], [Aus]

Gegenüber Wikipedia weist das Austria-Forum einige wesentliche Unterschiede auf. Offensichtlich ist die Beschränkung auf Themen mit Bezug zu Österreich. Darüber hinaus handelt es sich nicht um *ein* Lexikon sondern um eine Sammlung von ungefähr 40 Lexika zu verschiedenen Bereichen aus unterschiedlichen Zeiten. Qualitätsgeprüfte Beiträge sind vom Autor gezeichnet und können zitiert werden. Spezielle Inhalte, wie etwa die interaktive Bibliothek oder die Integration eines eigenen Community-Bereichs, sind weitere Hauptunterscheidungsmerkmale. [AFI10]

4.2 Technische Umsetzung des Austria-Forums

Mit dem Übergang von AEIOU zum Austria-Forum wurde das System aus mehreren Gründen auch in technologischer Sicht auf neue Beine gestellt. Eine Vielzahl von Problemen — wie die Rückwärts-Navigation mittels Browser-Funktion, Speichern von Seiten oder Indizierbarkeit durch Suchmaschinen — waren auf

der ursprünglichen AJAX-Plattform[1] nicht oder nur bedingt lösbar. Durch Analyse der Anforderungen und Evaluierung verschiedener Systeme kristallisierte sich *JSPWiki*[2] als vielversprechende Technologie heraus um die Idee des Austria-Forums möglichst gut umzusetzen. [Tra09]

Beim JSPWiki handelt es sich um eine frei verfügbare Open Source WikiWiki-Engine, die auf Java, Servlets und Java Server Pages (JSP) basiert. Dabei werden unter anderem folgende Punkte unterstützt, die für das Austria-Forum von Bedeutung sind:

- Auszeichnungen zur Strukturierung und Formatierung von Text
- Möglichkeit Dateien anzufügen
- Speicherung der Beiträge als Textdateien mit UTF-8 Unterstützung (keine Datenbank nötig)
- Unterstützung von Berechtigungskonzepten auf Wiki-, Seiten-, Benutzer- und Gruppenebene
- Einfache Schnittstelle zur Einbindung von Erweiterungen (PlugIns)
- Methodik um Konflikte durch simultane Bearbeitung eines Beitrags durch mehrere Anwender zu vermeiden

[Jsp09]

Die Speicherung der Wiki-Seiten als Textdateien im UTF-8 Format kommt dabei der Implementierung der Suche mit Hilfe von Lucene vor allem beim Prozess der Indexierung sehr entgegen.

4.3 Lucene im Austria-Forum

Klassifizierung

Ein wesentlicher Bestandteil von Foren ist die Suche nach Beiträgen. Eine gute Implementation stellt für die Anwender einen großen Gewinn dar. Die Umsetzung des Austria-Forums als JSPWiki legt eine Java-basierende Open-Source Suchapplikation nahe und Lucene ist hierfür das Werkzeug der Wahl.

Die einzelnen Wiki-Beiträge sind als Text-Dateien am Server gespeichert und bilden die Datenbasis. Somit kann die Suchanwendung nach Abschnitt 2.1 als *Open-Source Desktop-Suchmaschine* mit homogenen Quelldokumenten klassifiziert werden.

1 Asynchronous Java and XML
2 http://www.jspwiki.org

Integration

Der ursprünglichen Implementation der Such-Funktionalitäten liegt Lucene in Version 3.0.0 zugrunde. Der `LuceneSearchProvider` stellt dabei die zentrale Klasse zur Integration der Bibliothek ins Austria-Forum dar. Sowohl der Indexierungs- als auch der Suchprozess sind umgesetzt. Einige Teilanwendungen — wie die Bibliographie-Suche — bauen auf dieser bestehenden Integration auf und dürfen durch Erweiterungen und Adaptierungen der Such-Funktionalitäten nicht ins Wanken geraten.

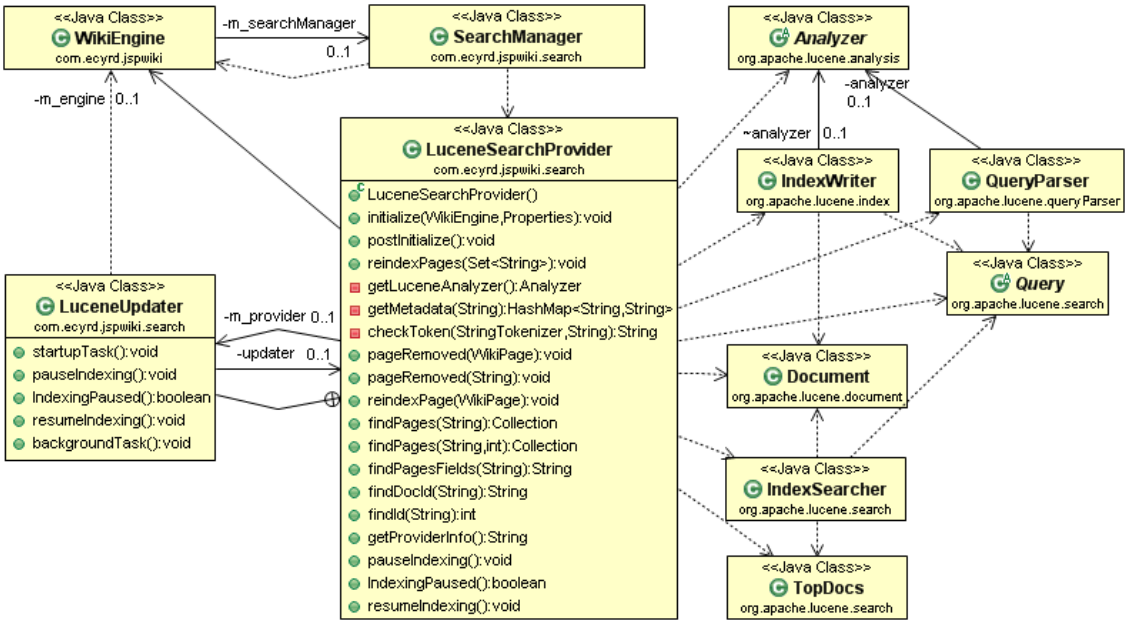

Abbildung 4.2: *Klassendiagramm zur Integration von Lucene ins Austria-Forum mit LuceneSearchProvider als zentraler Klasse*

Abbildung 4.2 illustriert die Anbindung der Klassen aus der Bibliothek an die *WikiEngine*, das Rückgrat des Austria-Forum. Die Darstellung ist stark vereinfacht und stellt nur die fundamentalen Klassen und Abhängigkeiten zur Schau. An dieser Stelle soll lediglich ein grober Überblick über die Integration von Lucene ins Austria-Forum vermittelt werden. Eine detailliertere Beschreibung erfolgt in den folgenden Kapiteln.

Die eingegebene Suchanfrage wird vom `LuceneSearchProvider` unter Verwendung der Klassen aus der Lucene-Bibliothek abgearbeitet. Die textuelle Suchanfrage wird dabei stets mittels `QueryParser` in einen Query transformiert. `IndexWriter` und `IndexSearcher` greifen auf den Index zu. Die Resultate werden als Objekt der Klasse `TopDocs` zurückgeliefert. Die Ergebnisseite bietet neben den Resultaten auch eine Maske zur Adaptierung der ursprünglichen Anfrage an, wie Abbildung 4.3 veranschaulicht. Der Index wird durch den `LuceneUpdater`, der

als eigenständiger Thread im Hintergrund läuft, erstellt beziehungsweise aktualisiert.

Weitere Details zur bestehenden Integration von Lucene im Austria-Forum werden an geeigneten Stellen in den nachfolgenden Kapiteln erörtert. Sie ermöglichen dort eine Gegenüberstellung von ursprünglicher und veränderter Funktionalität und erlauben somit eine Veranschaulichung der Effekte, die durch die Adaptierungen und Erweiterungen erzielt wurden.

Abbildung 4.3: *Ursprüngliche Ergebnisseite für den Suchbegriff „Tauern“ mit Eingabemöglichkeit zur Adaptierung der Anfrage*

5 Die Standard-Suche

Die Suche ist besonders bei Foren mitentscheidend für den Erfolg. Die nachfolgend beschriebenen Modifikationen sollen dem Austria-Forum zu noch besserem Anklang unter den Besuchern verhelfen. Das soll einerseits durch Erweiterungen im Funktionsumfang und andererseits durch Verbesserungen in der Bedienerfreundlichkeit erreicht werden.

Das vorliegende Kapitel thematisiert die spezifischen Anforderungen im Rahmen des Austria-Forum und die daraus resultierenden Vereinfachungen gegenüber den allgemeinen Betrachtungen aus den Kapiteln 2 und 3. Anhand der gewöhnlichen Such-Funktionalität werden auch die Indexierung und Benutzer-Interaktion beschrieben. Im Kontext der Standard-Suche stehen die qualitative Verbesserung der Suchergebnisse und die Aufwertung der Bedienerfreundlichkeit im Vordergrund. Diese Modifikationen wirken sich auch auf die Funktionserweiterungen aus, die in den Kapiteln 6 und 7 vorgestellt werden.

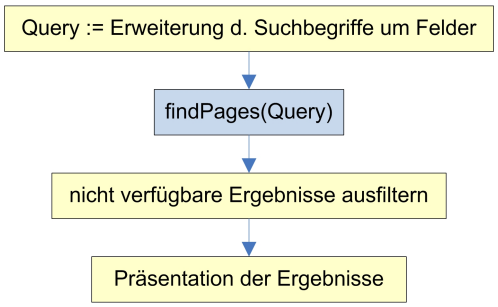

Abbildung 5.1: *Ursprünglicher Ablauf der Suche im Austria-Forum*

Abbildung 5.1 zeigt den schematischen Ablauf der ursprünglich implementierten Suche. Gelb hinterlegte Prozesse fanden in der als *Java Server Pages* implementierten Benutzerschnittstelle statt. Die Überführung der eingegebenen Suchanfrage in eine vom `QueryParser` verarbeitbare Zeichenkette erfolgte im Suchfenster. Dem `LuceneSearchProvider` wurde also statt der Suchanfrage bereits ein Query in Lucene-Syntax übergeben. Eine Filterung der Resultate vor der Präsentation ist aufgrund von unterschiedlichen Zugriffsberechtigungen nach wie vor nötig.

5.1 Indexierung

Die folgenden Fakten ergeben sich aus den Randbedingungen des Austria-Forums und vereinfachen den Prozess der Indexierung wesentlich.

Homogene Quelldokumente: Beiträge aus dem Forum sind als UTF-8 kodierte Text-Dateien am Web-Server abgelegt und werden durch den `PageManager` verwaltet. Die Text-Akquise beschränkt sich somit auf den Abruf der verwalteten Seiten und eine Konvertierung ist dabei nicht nötig.

Dokumentenspeicher: Die Quelldokumente sind lokal am Web-Server verfügbar. Im Vergleich zu Internet-Suchmaschinen sind die Datenbasis und somit auch die Anzahl der zu erwartenden Resultate wesentlich geringer. Eine Vorschau bei der Präsentation der Treffer erscheint somit verzichtbar. Auch die Bereitstellung der Resultate in einem vereinheitlichten Format stellt aufgrund der ohnehin homogenen Quelldokumente keine Anforderung dar. Somit ist ein separater Dokumentenspeicher im Zuge der Text-Akquise nicht zu berücksichtigen und kann entfallen.

Datenbasis: Beiträge im Austria-Forum sind in deutscher Sprache verfasst.[1] Dadurch wird die Text-Transformation vereinfacht und kann für eine Sprache optimiert werden. Eine allfällige Bestimmung der Sprache zur Steuerung der Textbereinigung oder Kanonisierung ist nicht erforderlich.

Analyse: Die vom `PageManager` verwalteten Seiten erlauben einfachen Zugriff auf einige Metadaten wie Titel, Beschreibung, Änderungsdatum oder Autor. Schlüsselwörter können bei der Erstellung eines Beitrags in definierten Tags angegeben werden und in separaten Feldern indexiert werden. Auf eine weiterführende strukturelle Analyse kann daher verzichtet werden. Die Reihung der Ergebnisse erfolgt innerhalb des Forums ausschließlich anhand Dokument-interner Daten. Eine Auswertung von Dokument-externen Daten — wie beispielsweise eine Link-Analyse — ist daher ebenfalls nicht notwendig.

Index-Größe: Der beschränkte Umfang der Datenbasis erlaubt die Handhabe der Index-Dateien auf einem einzelnen Rechner. Auch die Speicherung der Inhalte im Index ist mit den Ressourcen eines einzelnen Computers vereinbar. Dadurch ist die Bereitstellung von Vorschauen auch ohne separaten Dokumentenspeicher möglich. Weder eine Verteilung des Index noch dessen Komprimierung sind dafür erforderlich.

1 Die geringe Menge anderssprachiger Quelldokumente ist vernachlässigbar.

Aktualität: Da die Seiten des Forums vom `PageManager` verwaltet werden, kann die Indexierung an Ereignisse dieser Klasse gekoppelt werden. Somit kann die Aktualisierung des Index bei Bedarf — etwa beim Hinzufügen oder Ändern von Beiträgen — veranlasst werden.

5.1.1 Die Ausgangslage

Neuerstellung des Index

Die Beschreibung der Indexierung erfolgt anhand einer privaten Methode der `LuceneSearchProvider`-Klasse. Sie wird aufgerufen wenn keine Index-Dateien vorhanden sind. Das stellt also den Fall einer Neu-Indexierung dar, der nach dem Pseudocode aus Abbildung 5.2 implementiert ist.

```
doFullLuceneReindex()
 1  writer ← newIndexWriter(IndexDirectory, Analyzer, maxFieldLength);
 2  allPages ← PageManager.getAllPages();
 3  for each page in allPages
 4  do text ← PageManager.getPageText(page);
 5     luceneIndexPage(page, text, writer);
 6  allAttachments ← AttachmentManager.getAllAttachments();
 7  for each attachment in allAttachments
 8  do text ← AttachmentManager.getAttachmentContent(attachment);
 9     luceneIndexPage(attachment, text, writer);
10  writer.optimize();
11  writer.close();
```

Abbildung 5.2: *Pseudocode der Neu-Indexierung durch die LuceneSearchProvider-Klasse*

Der `IndexWriter` wird in Zeile 1 mit dem Verzeichnis des Index, dem `Analyzer` und einer Option für die maximale Feldlänge instanziiert. Dabei wurde zur Text-Transformation der `StandardAnalyzer` mit der englischen Standard-Stoppwortliste eingesetzt beziehungsweise erste Versuche mit dem `GermanAnalyzer` gewagt.

Durch die wertvolle Vorarbeit des `PageManager`-Objekts beschränkt sich die Text-Akquise auf den Abruf der Wiki-Seiten (2) und deren Inhalte (4). Die Methode `luceneIndexPage` (5) implementiert die Indexierung einer einzelnen Seite. Allfällige Anhänge eines Beitrags werden separat indexiert (6-9) und sind somit als

eigene Objekte suchbar. Dadurch werden sie auch unabhängig vom referenzierenden Beitrag als Ergebnis präsentiert und gereiht.

Während des Durchlaufs erstellt der `IndexWriter` einzelne Segment-Dateien,[1] die durch die Optimierung (10) zu einem einzelnen Segment verschmolzen werden. Abschließend wird der `IndexWriter` geschlossen (11) und damit die Änderung am Index bestätigt.

Darstellung 5.3 zeigt die Dateien des Index während der einzelnen Phasen. Die Datei *„segments.gen“* beinhaltet stets das Postfix der aktiven Segment-Datei, die Informationen über alle Segment-Dateien beinhaltet. Die *.fdt* und *.fdx*-Dateien beinhalten Daten und Referenzen für gespeicherte Felder.[2] Die *Compound Files (.cfs)* stellen Container für die segmentierten Dateien dar, die Felder, Term-Vokabular, Frequenzen und alle weiteren Daten des Index beinhalten.

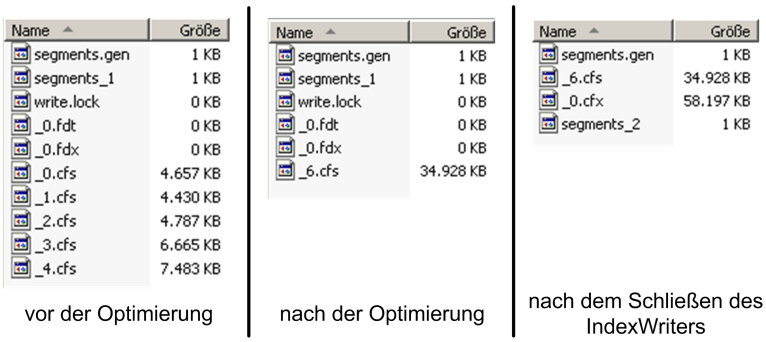

Abbildung 5.3: *Dateien des Lucene-Index vor und nach der Optimierung und bei geschlossenem IndexWriter*

Die Datei *„write.lock“* stellt sicher, dass nur ein `IndexWriter` schreibend auf den Index zugreift. Nach der Optimierung sind die Container der Segment-Dateien zu einem einzelnen Segment *„_6.cfs“* zusammengefasst. Beim Schließen des `IndexWriter` werden die Änderungen bestätigt, die gespeicherten Felder und Term-Vektoren zu einem *Compound File (.cfx)* zusammengefasst und die Lock-Datei gelöscht. [Theo6]

Die Quelldokumente und Anhänge werden sowohl bei der Neu-Indexierung als auch bei der Aktualisierung durch die selbe Methode indexiert. Der Pseudocode aus Abbildung 5.4 veranschaulicht diesen Algorithmus.

Zuerst wird eine Instanz der `Document`-Klasse erzeugt (1). Der Seitenname stellt den Identifikator der Seite dar und wird daher unverändert an das `Document`-

1 siehe Abschnitt „Segmentierung“ auf Seite 60
2 siehe Seite 59

```
luceneIndexPage(page, text, writer)
 1  doc ← newDocument();
 2  doc.add(newField("ID", page.name, Store, notAnalyzed));
 3  i ← 0;
 4  category ← getCategory(page.name, i);
 5  while category ≠ nil
 6  do doc.add(newField("category" + i, category.name, Store, notAnalyzed));
 7     i ← i + 1;
 8     category ← getCategory(page.name, i);
 9  doc.add(newField("content", page.text, Store, analyzed));
10  doc.add(newField("title", page.title, Store, analyzed));
11  doc.add(newField("author", page.author, Store, analyzed));
12  keywords ← getKeywords(page.text);
13  for each keyValue in keywords
14  do doc.add(newField(keyValue.key, keyValue.value, Store, analyzed));
15  attachments ← AttachmentManager.getAttachments(page);
16  for each attachment in attachments
17  do doc.add(newField("attachment", attachment.name, Store, analyzed));
18  writer.addDocument(doc);
```

Abbildung 5.4: *Pseudocode der Indexierung einer Wiki-Seite durch die LuceneSearchProvider-Klasse*

Objekt angehängt (2).[1] Die Optionen *Store* und *notAnalyzed* legen fest, wie der `IndexWriter` das Feld behandelt. In den Zeilen 3–8 werden Kategorie und allfällige Unterkategorien der Seite ermittelt und beigefügt. Die erste Ebene stammt dabei aus dem Austria-Forum. Unterkategorien stammen aus der hierarchischen Struktur des AEIOU. Die Tiefe der Kategorie bestimmt dabei den Identifikator. Auf diese Weise wird eine Suche innerhalb bestimmter Kategorien und Verzeichnistiefen für Beiträge ermöglicht und dabei die hierarchische Information aus AEIOU beibehalten.

In Zeile 9 wird der Inhalt des Beitrags an das Dokument angehängt. Im Gegensatz zu den bisherigen Feldern wird der Inhalt aufgrund der Option *analyzed* einer Text-Transformation unterzogen. Selbiges gilt für den Titel und Autor des Beitrags (10, 11). Schlüsselwörter und zugehörige Werte, die in einer bestimmten Notation im Beitrag vorliegen müssen, werden in den Zeilen 12–14 dem `Document` zugefügt. Die Namen allfälliger Anlagen werden im Feld „attachments“ akkumuliert (15–17). In Zeile 18 wird dem `IndexWriter` abschließend

1 Die Instanziierung des Feldes und das Hinzufügen zum Dokument werden aus Platzgründen in einer Zeile zusammengefasst.

das aufbereitete `Document`-Objekt zur Aufnahme in den Index übergeben.

Indexierte Felder

Der Index des Austria-Forums beinhaltet demnach folgende obligatorische Felder, die für jedes Dokument angelegt werden:

id: Der Dateiname des Beitrags beziehungsweise Anhangs ist zugleich der Identifikator des Dokuments. Er wird — als einziges Feld — gespeichert ohne davor einer Transformation zu unterliegen.

contents: Der gesamte Inhalt des Dokuments

name: Der Titel des Beitrags

attachment: Die Namen allfälliger Anhänge (Die Anhänge an sich werden als eigene Dokumente indexiert)

Darüber hinaus werden nachstehende Felder optional in den Index aufgenommen, sofern sie aus dem Dokument ermittelt werden können. Die Feldinhalte werden allesamt transformiert und gespeichert.

category*i*: Die Bezeichnungen der Kategorien werden aus der Verzeichnisstruktur ermittelt. Somit bleiben die hierarchischen Informationen aus AEIOU erhalten. Die Laufvariable *i* entspricht dabei der Verzeichnistiefe. Neben der Bezeichnung der *i*-ten Kategorie wird auch der gesamte Pfad vom Stammverzeichnis bis zur aktuellen Unterkategorie gespeichert. Das erlaubt eine Filterung nach Resultaten aus einer speziellen Unterkategorie, auch wenn namensgleiche Unterverzeichnisse in anderen Kategorien existieren.

autor: Der Name des Autors, sofern er spezifiziert ist.

Metadaten Allfällige Metadaten müssen in einer bestimmten Syntax im Beitrag enthalten sein.[1] Einige wichtige Vertreter davon sind:

suchbegriffe Schlagworte, auch als Schlüsselwörter bezeichnet, können dezidiert indexiert und — neben dem Titel des Beitrags ebenfalls standardmäßig — durchsucht werden.

geburtsjahr, **todesort**, **arbeitsorte** ... Diese Felder werden speziell bei der Suche in Biographien verwendet, die eine eigenständige Funktionalität darstellt und nicht Gegenstand dieser Arbeit ist.

1 als Name-Wertkonstrukte innerhalb eines Metadata-Tags: [{Metadata *key =' vaule(s) '*]}

Aktualisierung des Index

Änderungen der Datenbasis werden in einer geeigneten Datenstruktur protokolliert, die vom `LuceneUpdater` gelegentlich abgearbeitet wird. Dabei kommt bei Bedarf wiederum die Methode aus Abbildung 5.4 zum Einsatz, die auch bei der Neuerstellung des Index Anwendung findet.

5.1.2 Modifikationen

Lucene-Version

Zahlreiche Fehler der ursprünglich eingesetzten Version 3.0.0 von Lucene sind in der zum Zeitpunkt der Umsetzung aktuellen Version 3.0.2 behoben. Zusätzlich sind einige Adaptierungen der Applikationsschnittstelle und Optimierungen enthalten. Eine Auflistung aller Änderungen ist in der Dokumentation[1] ersichtlich. Die Aktualisierung von Lucene wurde durch Austausch der Bibliothek, die als Java-Archiv vorliegt, und Aktualisierung der Referenzen im Java-Projekt des Austria-Forums bewerkstelligt.

Text-Transformation

Die Wahl der passenden `Analyzer`-Klasse zur Text-Transformation ist ein entscheidender Erfolgsfaktor für die Implementation einer Suchapplikation mit Lucene. Ein wesentlicher Einflussfaktor ist dabei die Sprache der Quelldokumente. [MHG10]

Es ist offensichtlich, dass der ursprünglich eingesetzte `StandardAnalyzer` mit seiner englischen Stoppwortliste nicht die optimale Wahl für größtenteils deutschsprachige Quelldokumente war. Lucene bietet zwei mögliche Alternativen für deutsche Texte an, den `GermanAnalyzer` und den `SnowballAnalyzer` mit zwei alternativen Algorithmen zur Wortstammbildung.

1 http://lucene.apache.org/core/old_versioned_docs/versions/3_0_2/changes/Changes.html

GermanAnalyzer: Bei diesem Verfahren werden zunächst der Text zerlegt und die Token normalisiert. Danach erfolgt eine Überführung auf Kleinschreibung. Bei der nachfolgenden Text-Bereinigung kann entweder die inkludierte Stoppwortliste oder eine übergebene Alternative verwendet werden. Anschließend erfolgt die Wortstammbildung. Hierzu kann eine Liste mit Wörtern übermittelt werden, die von der Transformation ausgeschlossen und dadurch in ihrer ursprünglichen Form erhalten bleiben sollen.

Der Algorithmus wendet die Transformation nur auf Terme an, die ausschließlich aus Buchstaben bestehen. Zunächst werden Umlaute durch den jeweiligen Selbstlaut und einige Zeichenfolgen wie *sch* oder *ch* durch Sonderzeichen ersetzt. Danach wird das Ende des Terms beschnitten. Endungen wie *em, nd, er* sowie die Buchstaben *e, s, n* und *t* werden dabei rekursiv abgeschnitten. Nach einer Optimierung für Terme in Mehrzahl erfolgt die Rückersetzung der Sonderzeichen.[1]

Der SnowballAnalyzer: [Sno10] *Snowball* ist eine spezialisierte Sprache mit der Algorithmen zur Bildung von Wortstämmen realisiert werden können. Die Benennung ist ein Tribut an die Sprache *SNOBOL*, von der das Konzept zur Ablaufsteuerung durch Zeichenketten übernommen wurde. Damit lassen sich leistungsfähige Algorithmen zur Wortstammbildung — die *Stemmer* — implementieren.

Stemmer

Die Lucene-Implementation des `SnowballAnalyzer` stellt die üblichen Methoden zur Text-Zerlegung und -Bereinigung zur Verfügung. Für die Stammbildung stehen dabei einige Klassen bereit. Dazu gehören auch `GermanStemmer` und `German2Stemmer`. Deren Algorithmen ersetzen zunächst *ß* durch *ss*. Treten die Buchstaben *u* oder *y* zwischen zwei Selbstlauten auf, so werden sie durch ihre Großbuchstaben ersetzt und dadurch ein weiteres Abschneiden von Suffixen verhindert. Anschließend werden der Reihe nach die längsten der folgenden Endungen abgeschnitten:

- *em, ern, er, e, en, es* und s-Endungen wie *bs, ds, fs, gs, hs, ks, ls, ms, ns, ts*.
  Beispiele:
  äckern → *äck*, *ackers* → *acker*, *armes* → *arm*, *bedürfnissen* → *bedürfnis*
- *en, er, est* und st-Endungen wie *bst, dst, ...*
  Beispiele:
  derbsten → *derbst* (Schritt 1), und *derbst* → *derb* (Schritt 2)
- *end, ung, ig, ik, isch, lich, heit, keit*

1 Umlaute bleiben dabei als deren Stammvokale erhalten

Nach dem Beschneiden der Suffixe werden allfällige *U* und *Y* wieder durch ihre Kleinbuchstaben und Umlaute durch ihre Stammvokale ersetzt.[1]

Der `German2Stemmer` berücksichtigt darüber hinaus, dass die deutschen Umlaute auch oft als *ae, oe* beziehungsweise *ue* geschrieben werden. Sie werden im ersten Schritt durch *ä, ö* und *ü* ersetzt.[2] Dadurch werden Typen mit unterschiedlicher Schreibweise der Umlaute auf den selben Term transformiert.

StandardAnalyzer, GermanAnalyzer und SnowballAnalyzer sind *Stemming-Algorithmen*. Sie führen die Typen daher nicht auf ihre Lemmata sondern auf (fiktive) Stämme zurück!

Nr.	StandardAnalyzer	GermanAnalyzer	SnowballAnalyzer
1	und	src	und
2	src	imag	src
3	der	wissenssammlung	der
4	image	wien	imag
5	die	den	in
6	von	alt	die
7	im	osterreich	von
8	des	slideshowplugi	osterreich
9	wissenssammlungen	bild	im
10	mit	dem	ein
11	den	lef	des
12	das	class	wissenssaml
13	wien	an	mit
14	aus	aeiou	wien

Tabelle 5.1: *Gegenüberstellung der 14 häufigsten Index-Terme für Beiträge im Austria-Forum unter Verwendung des Standard- German- bzw. SnowballAnalyzer mit German2-Stemmer bei identischer Datenbasis*

Darstellung 5.1 verdeutlicht, dass die Verwendung von verschiedenen Algorithmen zu unterschiedlichen Index-Termen führt. Alle drei Beispiele implementieren Stemming, ihre Ergebnisse sind daher — oft fiktive — Wortstämme und im Allgemeinen nicht die tatsächlichen Wörterbuch-Formen der Typen.[3]

1 Die Beschreibung vermittelt das Prinzip und verzichtet dabei auf einige Implementations-Details.
2 Ausnahme: *ue* mit voran stehendem *q*
3 siehe Abschnitt 2.4.2 auf Seite 26

Ein empirischer Vergleich dieser Verfahren zeigte, dass der `SnowballAnalyzer` mit dem `German2Stemmer` die besten Resultate für die Suche in der Datenbasis des Austria-Forum liefert. Er ersetzt folglich den ursprünglich verwendeten `StandardAnalyzer`. Auf eine detaillierte Evaluierung mittels Präzision-Ausbeute-Diagramm wurde wegen des hohen Aufwandes zur Bestimmung relevanter Dokumente zu einer aussagekräftigen Zahl verschiedener Suchbegriffe verzichtet.

Durch die Adaptierung der Text-Transformation wurde die Qualität der Suche merklich verbessert. Besonders auffälig ist der Gewinn bei Wörtern mit Besonderheiten der deutschen Sprache, wie *ß* oder Umlauten. Auch das Abschneiden der sprachspezifischen Wortendungen führt häufig zu besseren Resultaten.

Die Tabelle 5.2 zeigt die transformierten Index-Terme und die Resultate für eine Suche nach dem Kurort Laßnitzhöhe. Die Schreibweise im Quelldokument ist dabei *„Laßnitzhöhe“*. Der `StandardAnalyzer` liefert nur Treffer wenn der Suchbegriff — abgesehen von Groß-/ Kleinschreibung — exakt der Zeichenfolge im Quelldokument entspricht.

	StandardAnalyzer	SnowballAnalyzer
Index-Term	laßnitzhöhe	lassnitzhoh
Treffer für	laßnitzhöhe	laßnitzhöhe lassnitzhöhe laßnitzhoehe lassnitzhoehe

Tabelle 5.2: *Gegenüberstellung der Treffer bei Suche nach dem Kurort "Laßnitzhöhe" unter Verwendung des Standard- bzw. SnowballAnalyzer mit German2-Stemmer*

Der für die deutsche Sprache entwickelte Algorithmus führt hingegen auch bei den unterschiedlichen Schreibweisen mit *ss* und *oe* zu Ergebnissen. Auch bei Wörtern ohne Umlauten erzielt der `SnowballAnalyzer` mit `German2Stemmer` aufgrund der Ausrichtung auf deutsche Endungen im Allgemeinen bessere Resultate als die ursprünglich eingesetzte Klasse.

5.2 Benutzer-Interaktion

5.2.1 Die Ausgangslage

Das Suchfenster in der rechten oberen Ecke[1] stellt einen omnipräsenten Einstiegspunkt in die Suche dar. Die Position und das intuitive Erscheinungsbild dieses Fensters sind den Besuchern des Austria-Forum bereits vertraut und wurden deshalb bewusst nahezu unverändert belassen.

Syntax der Suchanfrage

Die eingegebenen Begriffe wurden im Suchfenster um Feldnamen erweitert und in einer für Lucene verständlichen Syntax zur Weiterverarbeitung übergeben. Dabei waren zwei nahezu identische *Java Server Pages (JSP)* im Spiel, die nacheinander die selbe Anfrage zwei mal ausführten. Der Query wurde in seiner expandierten Form — also mitsamt der Feldnamen — zwischengespeichert um auf der Ergebnisseite eine Verfeinerung oder Änderung zu ermöglichen. Dabei musste die Modifikation der Suchanfrage durch den Anwender in Lucene-Syntax erfolgen.

Abbildung 5.5 zeigt die wenig benutzerfreundliche Darstellung des Query in Lucene-Syntax auf der Ergebnisseite.

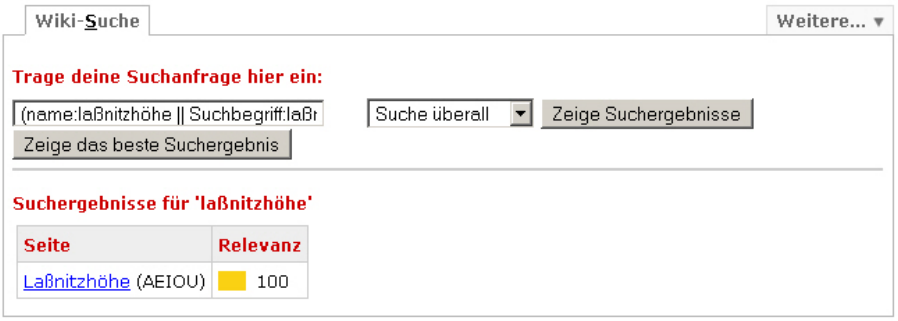

Abbildung 5.5: *Ursprüngliche Ergebnisseite für einen Treffer — Query in Lucene-Syntax und unverhältnismäßig kurzer Balken zur grafischen Veranschaulichung der Relevanz*

1 siehe Abbildungen 4.1 auf Seite 71 und 4.3 auf Seite 75

Grafische Anzeige der Relevanz

Die gelben Balken sollen die Relevanz der Treffer optisch widerspiegeln und somit einen Vergleich mehrerer Ergebnisse vereinfachen. Die Länge der Balken wurde dabei durch die Maximal- und Minimalwerte der dargestellten Resultate beeinflusst. Dadurch wurden, wie in den Darstellungen 5.5 und 5.6 deutlich erkenntlich ist, oftmals grafische Darstellungen präsentiert die nicht den tatsächlichen Verhältnismäßigkeiten entsprachen. Das beste Resultat wird dabei nach wie vor stets auf 100% skaliert.

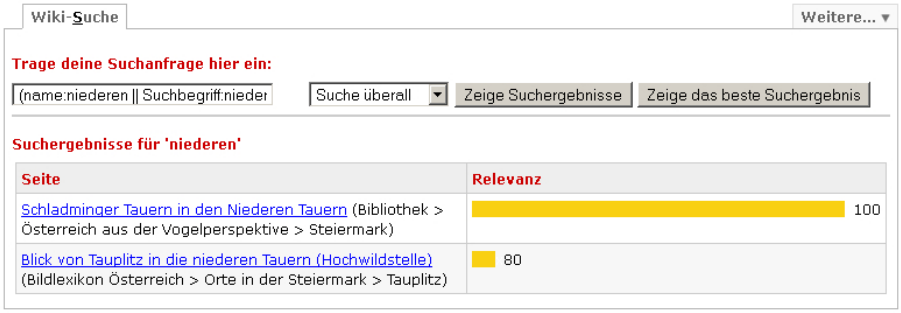

Abbildung 5.6: *Optische Darstellung der Dokument-Relevanz bei mehreren Resultaten — disproportionale Balkenlänge, beeinflusst durch Maximal- und Minimalwert der angeführten Treffer und Query in Lucene-Syntax*

5.2.2 Modifikationen

Syntax der Suchanfrage

Die im Suchfenster eingegebene Anfrage wird in ihrer ursprünglichen Form zwischengespeichert. Somit kann sie unverändert auf der Ergebnisseite zur Modifikation angeboten werden, wie Abbildung 5.7 zeigt. Dadurch bleibt dem Anwender die technische Schreibweise gänzlich verborgen und die Benutzerfreundlichkeit wird erhöht.

Bei einer Verfeinerung oder Änderung der Begriffe erfolgt eine neuerliche Erweiterung des Query entsprechend der vielfältigeren Optionen im erweiterten Suchfenster. Dazu werden die Begriffe und Optionen in einer Datenstruktur gesammelt und der Suchmethode übergeben. Die Erweiterung der Suchanfrage erfolgt nun also zentral in der `LuceneSearchProvider`-Klasse und nicht mehr im Suchfenster.

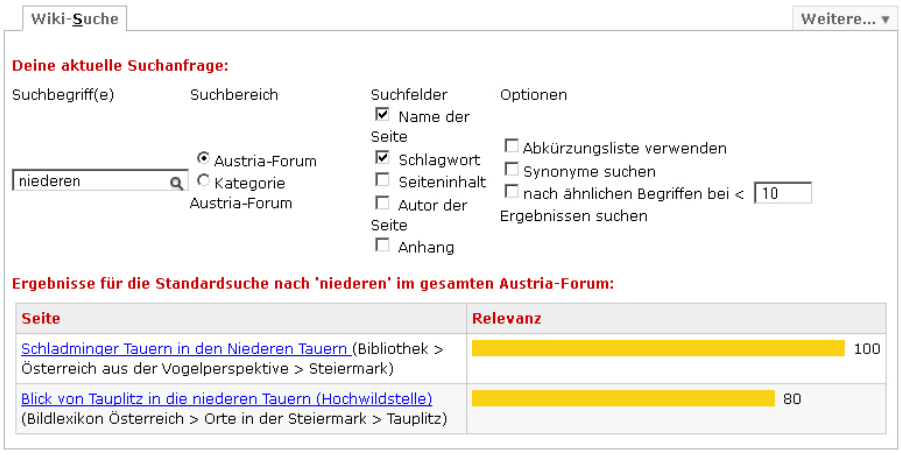

Abbildung 5.7: *Modifizierte Ergebnisseite — Suchanfrage in ihrer ursprünglichen Form und maßstäbliche Balkenanzeige der Relevanz*

Normalisierung der grafischen Relevanz-Anzeige

Neben der Bereinigung der obsoleten JSP-Seite wurde eine Normalisierung der grafischen Darstellung der Relevanz vorgenommen. Durch nicht sichtbare Referenz-Balken wird dabei eine gleichförmige Skalierung der Balken erreicht. Auch bei einem einzelnen Treffer wird der Balken für das (beste) Resultat auf die maximale Länge gestreckt, wie in Abbildung 5.8 zu sehen ist.

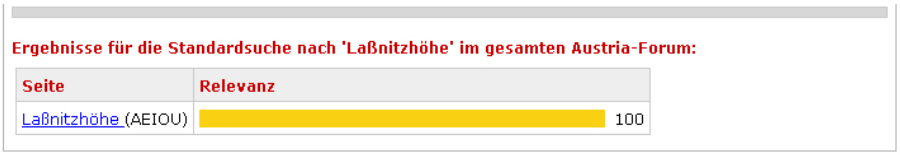

Abbildung 5.8: *Modifizierte Ergebnisseite mit konstanter Balkenlänge bei 100% Relevanz auch bei einem einzelnen Treffer, vgl. Abbildung 5.5*

Die Balkenlänge vermittelt nun auch über mehrere Suchanfragen hinweg einen vergleichbaren Eindruck über die Qualität eines Ergebnisses. Abbildung 5.7 bekundet im Vergleich mit Darstellung 5.6 die bessere Vergleichbarkeit bei mehreren Treffern. Die Relevanz-Balken werden dem Benutzer stets im gleichen Maßstab präsentiert. Dadurch gewinnt diese grafische Darstellung an Aussagekraft und bringt somit erhöhten Nutzen für den Anwender.

5.3 Standard-Einstellungen für die Suche

Wie bereits erwähnt, wurde das omnipräsente Suchfenster bewusst nahezu unverändert belassen. Hier kann der Benutzer angeben ob das gesamte Forum durchsucht werden soll oder die Resultate anhand der aktuellen Kategorie gefiltert werden sollen. Des weiteren kann festgelegt werden ob nur die Standard-Felder[1] oder auch der Volltext der verwalteten Seiten durchsucht werden sollen. Diese Einstellungen gelten gleichermaßen für das Suchfenster und die erweiterte Suchmaske auf der Ergebnisseite.

Abbildung 5.9: *Das omnipräsente Suchfenster — Einstiegspunkt für die Suche im Austria-Forum*

Die Voreinstellungen veranlassen eine Suche in *Titel und Schlüsselwörtern aller Beiträge* im Austria-Forum. Weitere Optionen — wie die Verwendung der Abkürzungsliste (siehe Seite 93) oder die Ähnlichkeitssuche (siehe Seite 94) — sind per se ebenfalls aktiviert.

Tabelle 5.3 fasst die Voreinstellungen zusammen. Sie können vom Anwender auf der Ergebnisseite jederzeit geändert werden und bleiben für die Dauer einer Sitzung erhalten.

Einstellung	Wert
Suchbereich	gesamtes Forum
Suchfelder	Titel, Schlüsselwörter (Schlagworte)
Abkürzungsliste	aktiviert
Synonymsuche	aktiviert
Ähnlichkeitssuche	bei weniger als 10 Treffern aktiviert

Tabelle 5.3: *Voreinstellungen für die Suche im Austria-Forum*

Obwohl einige Funktionserweiterungen per Voreinstellung auch bei der Standard-Suche aktiv sind, erscheint es zielführender sie im Rahmen der benutzerdefinierten Suche zu erläutern. Die Auswirkungen der zugehörigen Optionen werden ebenfalls in den nachstehenden Kapiteln erläutert.

1 das sind Titel und Schlüsselwörter der Beiträge

6 Benutzerdefinierte Suche

Ein Ziel der benutzerdefinierten Suche ist es, dem Anwender mehr Möglichkeiten zur Steuerung der Suche zu geben, ohne ihm dabei Wissen über die Syntax von Lucene-Queries abzuverlangen. Die Kontrolle über die erweiterten Funktionen ist ein zusätzliches Anliegen. Durch die intuitive Oberfläche sollen Besucher die Gelegenheit erhalten, steuernd in den Ablauf der Suche einzugreifen.

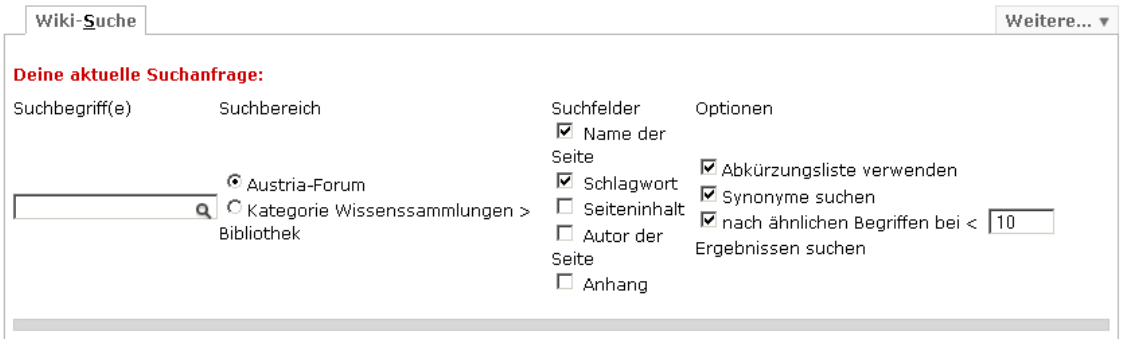

Abbildung 6.1: *Die erweiterte Suchmaske — Durch Wahl der Einstellungen können Ablauf und Verhalten der Suchfunktion gesteuert werden*

Die möglichen Einstellungen werden in Abbildung 6.1 illustriert. Auf der linken Seite kann die Suchanfrage kontrolliert oder modifiziert werden. Die Optionen auf der rechten Seite ermöglichen es verschiedene Erweiterungen ein- oder auszuschalten. Suchbereich und Suchfelder in der Mitte der Abbildung erlauben Kontrolle darüber *wo* gesucht wird.

6.1 Filterung und Wahl der Suchfelder

Befindet sich ein Besucher in einer bestimmten Kategorie des Austria-Forums — wie der Bibliothek, dem Bildlexikon oder einer Unterkategorie davon — so steht ihm die Möglichkeit offen, die Suche auf diesen aktuellen Teilbereich des Forums zu beschränken. Innerhalb dieses Teilbereichs[1] wird in den ausgewählten

1 beziehungsweise des gesamten Forums, wenn kein Bereich gewählt wurde

Feldern nach den Query-Termen gesucht. Dafür stehen prinzipiell alle indexierten Felder zur Verfügung, die Auswahl wurde jedoch auf die fünf bedeutendsten Felder beschränkt. Dadurch bleibt die Benutzeroberfläche übersichtlich und stellt trotzdem die wichtigsten Alternativen bereit.

Berücksichtigt werden die Einstellungen durch eine Erweiterung der Suchanfrage. Die Expansion wird dabei stets von der Klasse `LuceneSearchProvider` durchgeführt.

Query-Expansion

Zunächst werden die durch Leerzeichen getrennten Suchbegriffe eingeklammert. Die geklammerten Begriffe werden jedem gewählten Suchfeld angefügt und diese Query-Terme durch *oder*-Verknüpfungen[1] verbunden. Wenn die Suche auf eine spezielle Kategorie beschränkt wird, so werden die aneinandergefügten Teil-Queries erneut in Klammern gesetzt und mit der Kategorie *und*-verknüpft.[2] Dadurch wird eine Zeichenfolge geschaffen, die vom `QueryParser` korrekt interpretiert und verarbeitet wird. Die Klammern dienen zur Definition der korrekten Query-Architektur.

Das folgende Beispiel soll die Expansion des Query verdeutlichen. Der Bereich soll auf die Kategorie „Biographien“ beschränkt sein. Als Suchanfrage dient „Figl Leopold“. Dabei sollen nur der Name der Seite und die Schlagworte betrachtet werden. Der erweiterte Query stellt sich dann wie folgt dar:

```
(name:(figl leopold) ||suchbegriff:(figl leopold)) &&
normalizedcategory:Wissenssammlungen/Biographien
```

Die *und*-Verknüpfung mit der zweiten Zeile bewirkt dabei die Filterung. Durch die Verwendung der normalisierten Kategorie ist sichergestellt dass — auch bei gleichnamigen Unterkategorien in anderen Teilbereichen des Forums — die Resultate aus dem Bereich kommen, den der Besucher aktuell durchstöbert. Die Leerzeichen zwischen den Termen „figl“ und „leopold“ werden je Feld als *oder*-Verknüpfungen für dieses Feld interpretiert.

Durch diese Vorgehensweise können auch Operatoren wie „+“ oder „-“ in der Suchanfrage verwendet werden. Sie bleiben im erweiterten Query erhalten und werden vom Parser ebenfalls korrekt interpretiert.

1 Lucene-Syntax: ||
2 Lucene-Syntax: &&

6.2 Verwendung von Abkürzungen

Eine weitere Herausforderung stellt die Verwendung von Abkürzungen in der Suchanfrage dar. Das Ziel ist es beispielsweise für den Suchbegriff „St.Anton“ auch Dokumente zu finden die „Sankt Anton“ enthalten.

Um die Liste der Abkürzungen einfach warten zu können, wurde sie ebenfalls als Wiki-Seite im Austria-Forum implementiert. Im Unterschied zu den anderen Beiträgen wird diese Seite jedoch weder referenziert noch indexiert. Die Implementation dieser Funktionserweiterung erlaubt mehrere Abkürzungen für ein Wort und vice versa. Nach der beispielhaften Liste aus Abbildung 6.2 kann *Steiermark* sowohl als *stmk* wie auch als *st* abgekürzt werden. Die Abkürzung *st* steht gleichzeitig wiederum auch für *Sankt*.[1]

Abbildung 6.2: *Die Liste der Abkürzungen, zur einfachen Wartung ebenfalls als Wiki-Seite implementiert*

Query-Expansion

Wenn die Option *„Abkürzungsliste verwenden“* aktiviert ist, fließen die Einträge der Liste ebenfalls über eine Erweiterung des Query ein. Dazu werden die Suchbegriffe mit der Liste abgeglichen. Entspricht ein Suchbegriff einer definierten Abkürzung, so werden die zugeordneten Wörter entsprechend angefügt. Die Erweiterung der Suchbegriffe erfolgt *vor* der Transformation in Lucene-Syntax. Die aus Abschnitt 6.1 bekannte Logik bleibt dabei unverändert gültig. Somit wird in jedem Feld sowohl nach den vom Anwender eingegebenen Begriffen als auch nach den Entsprechungen aus der Abkürzungsliste gesucht.

Ein Beispiel mit Verwendung einer Abkürzung sowie eines Operators verdeutlicht die Expansion des Query. Wird bei einer Suche nach „schöne +stmk“ die Abkürzungsliste verwendet, so sieht der erweiterte Query wie folgt aus:

```
name:(schöne +(stmk || steiermark) ) ||
suchbegriff:(schöne +(stmk || steiermark) )
```

1 Die Groß/-Kleinschreibung ist dabei irrelevant.

Durch die Klammern um die *oder*-Verknüpfung aus dem ursprünglichen Begriff „stmk“ und seiner Entsprechung „steiermark“ aus der Abkürzungsliste bleibt der voran stehende „+“ Operator für den Klammerausdruck erhalten. Resultate müssen also zwingend entweder „stmk“ oder „steiermark“ enthalten. Der Begriff „schöne“ bleibt dabei als *oder*-Verknüpfung optional.

6.3 Ähnlichkeitssuche

Die Stammbildung bei der Indexierung und Suche erhöht die Trefferwahrscheinlichkeit dadurch, dass verschiedene Wortformen auf den selben Stamm reduziert werden. Sie zeigt ihre Wirkung also bei unterschiedlichen Wortformen die dabei jedoch korrekt geschrieben sein müssen. Diese Text-Transformation bringt im Allgemeinen keinen Gewinn bei möglichen Tipp- oder Schreibfehlern.

Ziel

Das Ansinnen der Ähnlichkeitssuche ist es, auch in solchen Fällen potenziell relevante Ergebnisse zu returnieren. Das bedeutet eine Ausweitung der Suche auf orthografisch ähnliche Begriffe. Geht man davon aus, dass eine größere Anzahl von Treffern auf eine korrekte Schreibweise der Suchanfrage hindeutet, so ist eine Suche nach ähnlichen Begriffen vor allem bei einer geringen Trefferquote der ursprünglichen Anfrage interessant. Die Ähnlichkeitssuche wurde daher der Suche nach den eingegebenen Begriffen nachgelagert. Diese Erweiterung kommt — sofern aktiviert — nur dann zum Tragen, wenn die Trefferquote der ursprünglichen Anfrage unter einer gewissen Schranke liegt. Die Voreinstellung liegt dabei bei zehn Resultaten und kann vom Benutzer angepasst werden.

Performanz

Ein großer Teil der Suchanfragen liefert mehrere Resultate und liegt daher über der Schranke. In diesen Fällen wird nur *eine* Suchanfrage durchgeführt und die Performanz bleibt unbeeinflusst.[1] Bei wenigen Resultaten erscheint eine verlängerte Suchzeit akzeptabel, wenn dadurch auch bei Tippfehlern in Frage kommende Ergebnisse präsentiert werden. Die längere Suchdauer kommt einerseits durch die doppelte Abarbeitung und andererseits durch die Methodik an sich zustande.

Bei einer Suche nach (exakten) Treffern genügt eine Abfrage der Postings zu diesem Term im invertierten Index. Bei einer Suche nach ähnlichen Begriffen muss hingegen *das gesamte Vokabular* betrachtet und für jeden Term im Index die Ähnlichkeit zum Suchterm errechnet werden. [MHG10] Aus diesem Grund wurde von einer generellen Erweiterung der Suchanfrage auf ähnliche Begriffe

1 Die Ergebnisse für orthografisch ähnliche Wörter würden gegenüber exakten Term-Treffern ohnehin auf hinteren Rängen gereiht und vom Anwender kaum verfolgt werden.

Abstand genommen und die zuvor genannte untere Schranke für die Ähnlichkeitssuche implementiert.

Query-Expansion

Lucene stellt für diesen Anwendungsfall einen `FuzzyQuerry` bereit. Durch Anfügen der Tilde an einen Suchbegriff wird ein solches Objekt vom `QueryParser` erzeugt. Die Suchanfrage wird derart erweitert, dass jeder Begriff neben seiner ursprünglichen Form ein zweites Mal mit angefügter Tilde vorkommt, die Anzahl der Begriffe also verdoppelt wird. Unveränderte Suchworte werden dabei quadratisch gewichtet, wohingegen Ähnlichkeits-Terme weiter abgeschwächt werden. Zusätzlich wird eine minimale Ähnlichkeit zwischen Index- und Suchterm verlangt. Ein Beispiel verdeutlicht wieder die Expansion.

```
name:(tauren^2 tauren~0.6^0.5) || suchbegriff:(tauren^2 tauren~0.6^0.5)
```

Der ursprüngliche Begriff „tauren“ wird quadratisch gewichtet. Zusätzlich wird mit der zugefügten Zeichenfolge auch nach ähnlichen Begriffen mit einer minimalen Ähnlichkeit von 0.6 gesucht und diese Treffer durch die Wurzel-Funktion schwächer gewichtet. Die Werte für die Gewichtung von unverändertem (^2) und und Ähnlichkeits-Term (^0.5) sowie die minimale Ähnlichkeit (0.6) wurden empirisch ermittelt.

Da die Erweiterungen vor der Expansion aus Abschnitt 6.1 durchgeführt werden, wird wiederum in jedem Feld sowohl nach den vom Anwender eingegebenen Begriffen als auch nach den ähnlichen Begriffen gesucht. Auf diese Weise werden nun beispielsweise auch Seiten mit „Tauern“ in Titel oder Schlagworten aufgrund der Ähnlichkeit zu „tauren“ als potenzielle Treffer zurück geliefert. Die minimale Ähnlichkeit ist dabei wie folgt definiert:

$$\text{Ähnlichkeit} = 1 - \frac{\text{LevenshteinDistanz}}{min(\text{IndexTermLänge}, \text{QueryTermLänge})}$$

Die Levenshtein-Distanz ist ein Maß für die Ähnlichkeit zweier Zeichenfolgen. Sie wird als Anzahl der nötigen Zeichenveränderungen[1] zur Überführung von Index- zu Query-Term ermittelt. [MHG10]

Abbildung 6.3 veranschaulicht den Ablauf der modifizierten Suche. Gelb hinterlegte Teile finden dabei in den Java Server Pages der Benutzerschnittstelle platz. Der bläuliche Mittelteil ist im `LuceneSearchProvider`, dem zentralen Objekt zur Integration von Lucene ins Austria-Forum, implementiert.

1 Zeichen einfügen, löschen oder ersetzen

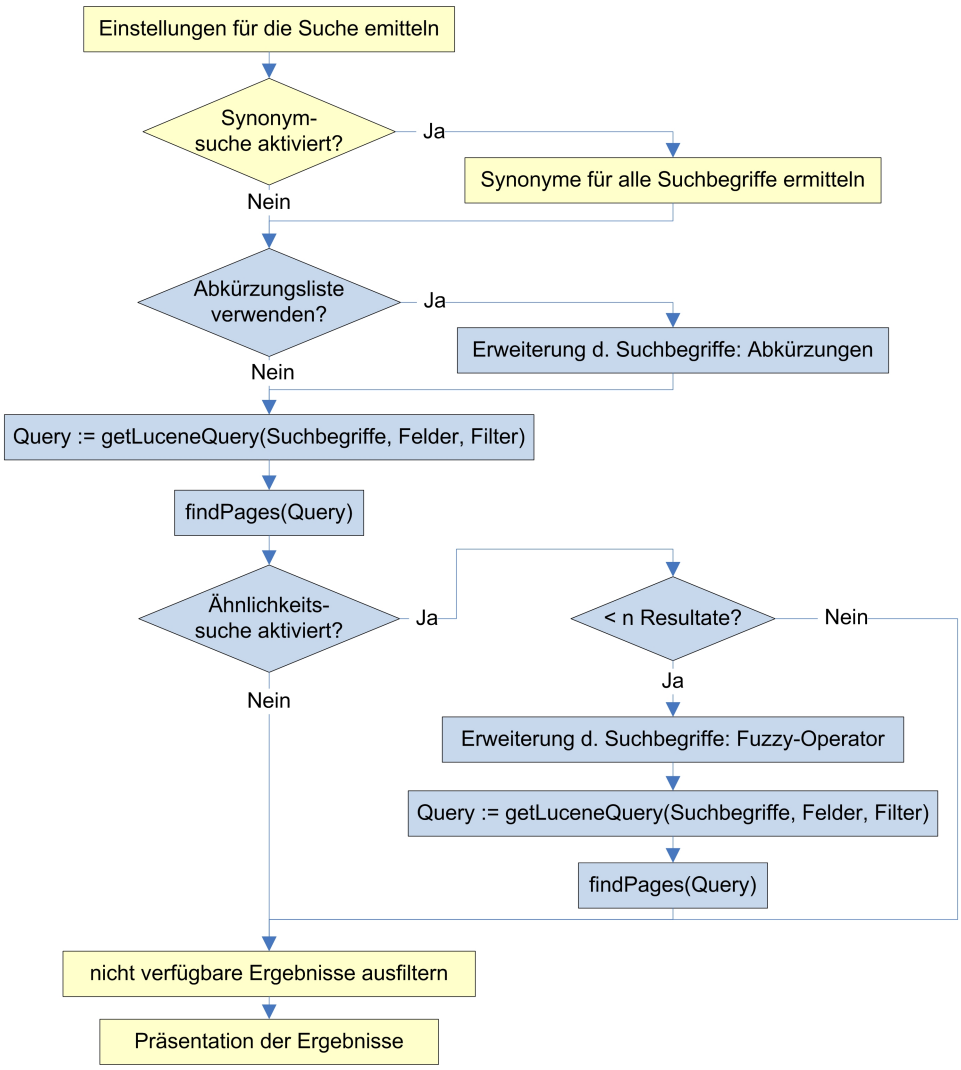

Abbildung 6.3: *Ablaufdiagramm der modifizierten Suche mit den erweiterten Funktionalitäten*
Vergleiche Abbildung 5.1 auf Seite 77

Die Synonymsuche, die ebenfalls im Flussdiagramm dargestellt ist, kann auch durch den Anwender aktiviert beziehungsweise deaktiviert werden. Sie stellt eine abgeschlossene Thematik dar und funktioniert unabhängig von der text-basierten Suche im Austria-Forum. Des weiteren ist sie losgelöst von Lucene, bedient sich also nicht des Indexes. Ihr sei daher ein eigenes Kapitel gewidmet.

7 Die Synonymsuche

Die Suche soll dem Anwender helfen möglichst komfortabel an gewünschte Informationen zu gelangen beziehungsweise relevante Inhalte zu entdecken. Die Methoden der vorangegangenen Kapitel zielen darauf ab, für eine vom Benutzer *vorgegebene Suchanfrage* bestmögliche Ergebnisse im Index aufzuspüren. In der Praxis hat sich eine unterstützende Herangehensweise bewährt, die den Hebel bei der *Neuformulierung der Suchanfrage* ansetzt.

Die Synonymsuche beeinflusst nicht die Resultate für eine gegebene Suchanfrage, sondern suggeriert eine alternative Formulierung durch Bereitstellung von sinnverwandten Begriffen!

Gegenstand dieses Kapitels ist also die *Suche nach Synonymen* für die eingegebenen Suchbegriffe um den Benutzer im Bedarfsfall bei einer Umformulierung der Anfrage zu unterstützen. Das ist nicht zu verwechseln mit einer automatischen Suche im Index nach Dokumenten, die zu den Suchbegriffen gleichbedeutende Wörter enthalten!

„Meinten Sie…?“

Oftmals bringt die Verwendung von bedeutungsgleichen Worten oder Begriffen von ähnlicher Bedeutung — den Synonymen — in der Anfrage unterschiedliche Ergebnisse hervor. Die Umformulierung der Suchanfrage kann Verbesserungen erzielen, wenn die Resultate für die ursprüngliche Anfrage nicht zufriedenstellend sind. Dabei ist es sehr hilfreich eine Auswahl von Synonymen zu den ursprünglichen Suchbegriffen anzubieten. Viele moderne Suchmaschinen offerieren solche Wörter auf der Ergebnisseite etwa unter einer Rubrik *„Meinten Sie vielleicht…?“* oder *„Versuchen Sie auch…“* und ersparen dem Benutzer dadurch separate Abstecher in diverse Thesauri.

Doch wie findet man Synonyme zu einem Begriff? Die Lösung hierfür bieten Strukturen, die neben den Wörtern auch die semantischen Relationen zwischen ihnen abbilden, sogenannte *lexikalisch–semantische Wortnetze*.

7.1 Begriffsbestimmung

Vorab seien einige Begriffe erklärend zusammengefasst.

Wortnetze — Lexikalisch–semantische Wortnetze fassen Begrifflichkeiten als *Konzepte* auf und setzen sie durch Verknüpfungen in Beziehung zueinander. Dabei kommen diverse Relationen wie *Teil* ↔ *Ganzes* oder *Oberbegriff* ↔ *Unterbegriff* in Frage. Dadurch werden sinngemäße Verbindungen zwischen orthografisch unterschiedlichen Wörtern abgebildet.

WordNet® — Diese lexikalische Datenbank wurde von der Universität Princeton[1] entwickelt und gilt gemeinhin als Mutter der lexikalisch–semantischen Wortnetze. Haupt-, Zeit-, Eigenschafts- und Umstandswörter der englischen Sprache werden darin gruppiert und zueinander in Relationen gesetzt.

Holonym — Ein Holonym bezeichnet ein Ganzes. Beispiele hierfür sind etwa *Hand* oder *Sessel*.

Meronym — Teile eines Ganzen werden als Meronyme bezeichnet. So sind *Finger* Teile einer Hand und *Lehne, Sitzfläche* Teile eines Sessels. Holonym und Meronym bilden also die Relation *Teil* ↔ *Ganzes* ab.

Hyperonym — Ein Hyperonym ist ein Oberbegriff. *Sitzmöbel* ist beispielsweise ein übergeordneter Begriff zu *Sessel, Stuhl, Bank, ...* und hat somit größeren Bedeutungsumfang.

Hyponym — Der Gegenspieler zum Hyperonym ist der Unterbegriff. Nach obigem Beispiel sind also *Sessel, Stuhl* und *Bank* Hyponyme vom Begriff *Sitzmöbel*, der seinerseits wieder ein Hyponym von *Möbel* ist. Wörter stellen also häufig eine Teilmenge eines Oberbegriffs dar und sind gleichzeitig auch Hyperonym für weitere Begriffe.

Durch diese Relationen wird eine hierarchische Struktur der Begrifflichkeiten definiert: Jeder Stuhl ist ein Sitzmöbel und gehört somit zu den Möbeln. Der Umkehrschluss ist dabei nicht zulässig: Nicht alles was als Möbel bezeichnet werden kann ist auch ein Stuhl.

Synonym — Verschiedene Wörter, die bedeutungsgleich oder bedeutungsähnlich sind, können gegeneinander ausgetauscht werden ohne den Sinn des Textes wesentlich zu verändern. Synonyme werden daher auch als *Ersetzungsworte* bezeichnet.

1 http://wordnet.princeton.edu/

Synset — Ein Synset fasst verschiedene Konzepte einer *Lesart* — also lexikalische Einheiten gleicher Bedeutung — zusammen und gruppiert somit synonyme Wörter zu einer Einheit.

Abbildung 7.1 veranschaulicht die Gruppierungen und Verknüpfungen in einem lexikalisch–semantischen Wortnetz anhand eines kleinen Ausschnitts aus dem GermaNet. Synonyme sind in Synsets zusammengefasst und diese Einheiten sind durch semantische und lexikalische Beziehungen miteinander verwoben. Eine Hyponymie ist die Relation von Ober- zu Unterbegriff, stellt also eine Unterordnung — eine Subordination — dar. Zugleich existiert zwischen den selben Synsets eine Hyperonymie — oder Superordination — in die Gegenrichtung.[1] Begriffe von gegensätzlicher Bedeutung werden durch Antonymien in Relation zueinander gebracht.

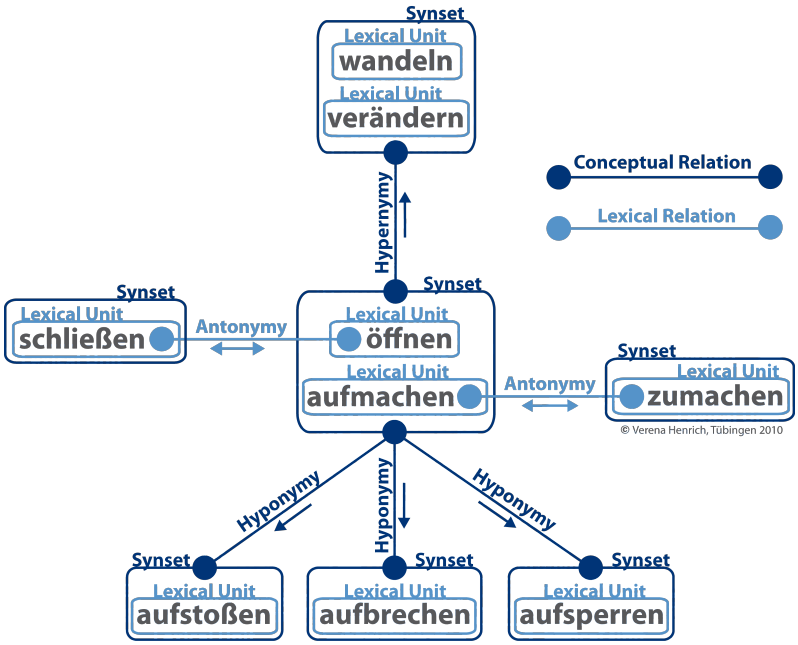

Abbildung 7.1: *Synsets und die verknüpfenden Relationen in einem Wortnetz anhand eines Ausschnitts aus GermaNet*
Quelle: [Gero9, Download-Bereich]

Durch die Verwendung eines Wortnetzes ist es demnach möglich dem Benutzer neben Synonymen beispielsweise auch Ober- und Unterbegriffe zur Umformulierung seiner Anfrage anzubieten. Da Hyperonyme und Hyponyme jedoch die Bedeutung erweitern beziehungsweise einengen, beschränkt sich die Funktionserweiterung im Rahmen des Austria-Forums auf die Darbietung von Synonymen um bedeutungsgleiche Alternativen zur ursprünglichen Suchanfrage zu suggerieren.

1 Die Abbildung zeigt zur besseren Übersichtlichkeit jeweils nur eine dieser Relationen.

7.2 Was ist GermaNet?

GermaNet ist ein lexikalisch–semantisches Wortnetz, das Hauptwörter, Verben und Adjektive der deutschen Sprache semantisch gruppiert. Es ähnelt dabei dem englischen WordNet® und kann als Online-Thesaurus betrachtet werden.

Entwickelt wurde dieses Wortnetz ab 1997 in mehreren Projekten des Seminars für Sprachwissenschaft (SfS) an der Universität Tübingen, welches auch die Wartung betreibt. GermaNet wurde auch in das mehrsprachige *EuroWordNet* aufgenommen. Die technische Umsetzung basiert weitestgehend auf dem Wortnetz der Princeton Universität, von dem Datenbankformat und Code übernommen wurden. Zur Handhabbarkeit von Besonderheiten der deutschen Sprache wurden geringfügige Modifikationen des Quellcodes vorgenommen. [Ger09]

> *„GermaNet ist aus verschiedenen lexikographischen Quellen [...] von Hand aufgebaut worden. In GermaNet sind die bedeutungstragenden Kategorien der Nomina, Verben und Adjektive modelliert. Zentrales Repräsentationskonzept ist das Synset, welches die Synonymenmenge eines gegebenen Konzeptes bereitstellt, z.B. {Streichholz, Zündholz}, {fleißig,eifrig, emsig, tüchtig} und {vergeben, verzeihen}.“* [LK07]

Folgende Zahlen spiegeln den aktuellen Umfang von GermaNet wider:

Charakteristikum	Größenordnung
Synsets	$\sim$ 70000
lexikalische Einheiten	$\sim$ 93500
konzeptionelle Relationen	$\sim$ 82000
lexikalische Relationen	$\sim$ 3600

Tabelle 7.1: *Aktueller Umfang des deutschen Wortnetzes GermaNet*
Datenquelle: [Ger09]

Die Summe der Beziehungen eines Wortes zu anderen Begrifflichkeiten beschreibt die Bedeutung dieses Wortes. In GermaNet werden dabei zwei verschiedene Typen von Relationen unterschieden, wie auch in Abbildung 7.1 ersichtlich ist.

Lexikalische Relationen stellen bidirektionale Beziehungen zwischen Wortbedeutungen her. Beispielhaft dafür sind die Bedeutungsgleichheit (Synonymie) innerhalb eines Synsets, oder die Gegenteiligkeit (Antonymie) zwischen verschiedenen Synsets.

Konzeptionelle Relationen bestehen zwischen Konzepten und gelten somit für alle lexikalischen Einheiten eines Synsets. Beispiele dafür sind etwa die bereits erwähnte Hyponymie und Hyperonymie oder auch Implikation und Kausalität.

[LK07]

Struktur

GermaNet wird durch eine Datenbank repräsentiert und im Allgemeinen als Bündel von XML-Dateien ausgeliefert. Darin sind die Nomen, Verben und Adjektive nach verschiedenen Klassen unterteilt und in Synsets zusammengefasst über mehreren Dateien verteilt. Die Relationen zwischen den Synsets sind ebenfalls in einer XML-Datei abgelegt. Abbildung 7.2 zeigt einen Ausschnitt aus einer XML-Datei mit Adjektiven für ein spezifisches Synset.

```
- <synset id="s2255" category="adj">
  - <lexUnit id="l3469" sense="1" source="core" namedEntity="no" artificial="no" styleMarking="no">
      <orthForm>endlos</orthForm>
    </lexUnit>
  - <lexUnit id="l3470" sense="1" source="core" namedEntity="no" artificial="no" styleMarking="no">
      <orthForm>unendlich</orthForm>
    </lexUnit>
    <paraphrase>ohne räumliches Ende</paraphrase>
  </synset>
```

Abbildung 7.2: *Beispiel eines GermaNet-Synsets in der XML-Struktur*

Darstellung 7.3 veranschaulicht beispielhaft die Abbildung einer Relation für das Synset aus Abbildung 7.2. Das Präfix „con“ in kennzeichnet dabei eine konzeptionelle Relation. Lexikalische Beziehungen werden durch das Präfix „lex“ gekennzeichnet.

```
<con_rel name="hyperonymy" from="s2255" to="s2247" dir="revert" inv="hyponymy" />
```

Abbildung 7.3: *Notation einer Relation zwischen Synsets in GermaNet*

Tabelle 7.2 fasst die implementierten Relationen zusammen. Neben den aus Abschnitt 7.1 bekannten Beziehungen werden auch Implikationen[1] und Kausalitäten[2] abgebildet. Pertonymie und Partizip dienen dazu, hauptwörtlich gebrauchte Verben mit ihrem ursprünglichen Zeitwort oder verschiedene Verbformen und Mittelwörter untereinander in Beziehung zu setzen. Über Assoziationen können relativ lose Verbindungen wie *Schließvorrichtung* ↔ *schließen* hergestellt werden. [Ger09]

Wie kann nun ein solches lexikalisch-semantisches Wortnetz an eine Suchapplikation angebunden und nutzbar gemacht werden?

1 „schnarchen“ impliziert „schlafen“
2 „zerschleißen“ und „zerstört“ hängen kausal zusammen

Relation	Typ	Inverse Relation
Synonymie	lexikalisch	Synonymie
Antonymie	lexikalisch	Antonymie
Hyperonymie	konzeptionell	Hyponymie
Hyponomie	konzeptionell	Hyperonomie
Meronymie	konzeptionell	Holonymie
Holonymie	konzeptionell	Meronymie
Implikation	konzeptionell	—
Kausalität	konzeptionell	—
Pertonymie	lexikalisch	—
Partizip	lexikalisch	—
Assoziation	konzeptionell	—

Tabelle 7.2: *Relationen im GermaNet*
modifizierte Darstellung von [Gero9, Struktur-Relationen]

7.3 Germanet-Integration

Lucene bietet für das WordNet® Programme wie *Syns2Index* oder *SynLookup* an, mit denen aus dem lexikalisch-semantischen Wortnetz ein Lucene-Index erstellt werden und darin nach Synonymen gesucht werden kann. [MHG10] Für das englischsprachige WordNet® gibt es also die Möglichkeit eine Abbildung als Lucene-Index zu erstellen und in diesem Index zu suchen.

GermaNet stellt mehrere Werkzeuge bereit um die Daten nutzbar zu machen. Neben Anwendungen wie *GermaNet-Explorer* und *GernEdiT* zur Visualisierung und Bearbeitung der Daten stehen auch Scripts zur Konvertierung und vor allem Applikationsschnittstellen in Java und Perl zur Verfügung.[1]

Die Java-Anwendungsschnittstelle ist das Mittel der Wahl zur Integration von GermaNet ins Austria-Forum. Sie beinhaltet mehrere Klassen, von denen vor allem `GermaNet` und `Synset` hervorzuheben sind. Abbildung 7.4 zeigt diese wesentlichen Klassen und deren Anbindung an die Suchfunktionalität im Austria-Forum.

GermaNet-Klasse

Die `GermaNet` Klasse implementiert Methoden um auf Synsets und lexikalische Einheiten zuzugreifen. Bei der Instanziierung kann ein Verzeichnis mit den XML Nutzdaten angegeben werden. Diese Klasse ermöglicht den Zugriff auf das Wortnetz zur Suche und Auswertung. Eine Modifikation der zugrunde liegenden GermaNet-Daten wird dabei nicht unterstützt.

1 siehe `http://www.sfs.uni-tuebingen.de/lsd/tools.shtml`

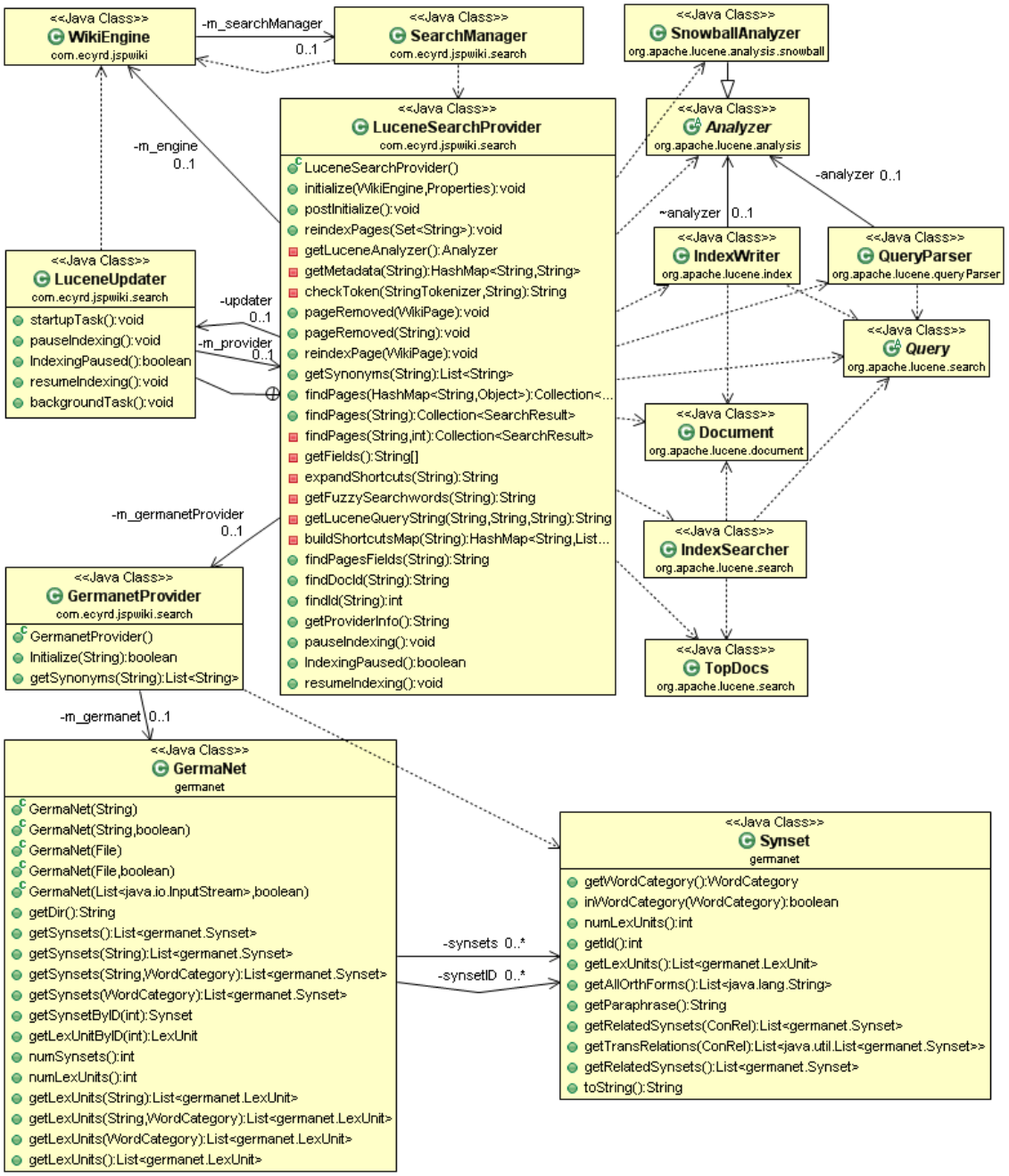

Abbildung 7.4: *Klassendiagramm der erweiterten Indexierung und Suche im Austria-Forum mit Anbindung des GermaNet*

Synset-Klasse

Bedeutungsgleiche lexikalische Einheiten werden über die `Synset` Klasse adressiert. Neben der Wortform — also Haupt- Zeit- oder Eigenschaftswort — können so auch synonyme Wörter, alternative Schreibweisen oder erklärende Phrasen abgerufen werden.

Die Synonymsuche ist mittels GermaNet-Applikationsschnittstelle unabhängig von Lucene realisierbar. Eine sprachliche Übereinstimmung zwischen Quelldokumenten, Suche im Index und Synonymsuche ist jedoch zweckmäßig.

Die GermaNet-Daten werden also im Austria-Forum unmittelbar über die Applikationsschnittstelle genutzt. Es wird kein Lucene-Index für die Synonyme erstellt und die Synonymsuche ist daher prinzipiell unabhängig von Lucene realisierbar. Es besteht jedoch ein sinngemäßer Zusammenhang zwischen diesen beiden Komponenten: Sowohl die Synonymsuche als auch die Suche im Index sind von der Sprache der Quelldokumente abhängig.

Für eine spezifische Sprache der Quelldokumente ist ein geeigneter, sprachabhängiger `Analyzer` von entscheidender Bedeutung für die Qualität der Suche. Gleichzeitig erscheint auch die Darbietung von Synonymen nur dann als sinnvoll, wenn sie sich der selben Sprache bedient. Bei einsprachigen Datenbasen — wie das beim Austria-Forum der Fall ist — sollten folglich die Spracheinstellungen bei der Text-Transformation und Synonymsuche aufeinander abgestimmt sein. Aus diesem Grund wurde die technisch unabhängige Synonymsuche ebenfalls unter Verwendung des `LuceneSearchProvider` aus den vorangegangenen Kapiteln implementiert. Somit ist sichergestellt, dass bei einer eventuellen Ablöse von Lucene auch über die Synonymsuche nachgedacht werden muss. Das verhindert ein Auseinanderdriften dieser Komponenten bei späteren Modifikationen.

Zu diesem Zweck wurden der `LuceneSearchProvider` und die zugrunde liegende Schnittstellendefinition um eine Methode zum Abruf der Synonyme erweitert. Darüber hinaus wurde die Klasse `GermanetProvider` implementiert, die den Zugriff auf die GermaNet-Anwendungsschnittstelle kapselt. Das vereinfachte Klassendiagramm in Abbildung 7.4 veranschaulicht diese Integration.

Wenn die Synonymsuche aktiviert ist, werden durch die Methode `getSynonyms` des `LuceneSearchProvider` zunächst Alternativen zu den eingegebenen Suchbegriffen ermittelt. Dieser Ablauf wurde im Flussdiagramm 6.3 auf Seite 96 skizziert. Wie bereits erwähnt, bleibt die Suche im Index davon unberührt, die Anfrage also unverändert.

Allfällig vorhandene Synonyme werden dem Benutzer auf der Ergebnisseite zwischen der erweiterten Suchmaske und den Ergebnissen präsentiert, wie Abbildung 7.5 zeigt. Die Synonyme wurden dabei als Verknüpfungen realisiert. Bei einem einzelnen Suchbegriff kann der Anwender durch einen Klick auf ein Wort aus den alternativen Begriffen eine neue Suche nach dem Synonym starten, ohne dieses manuell eintippen zu müssen. Auf diese Weise wird eine komfortable Neuformulierung der Suchanfrage ermöglicht.

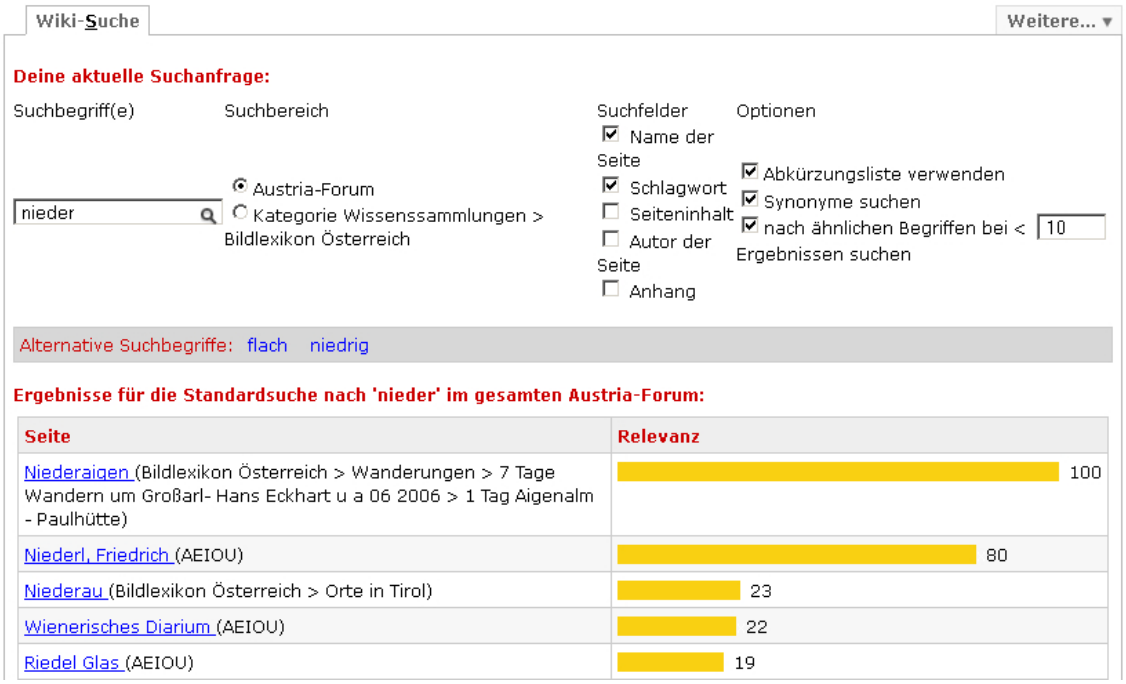

Abbildung 7.5: *Modifizierte Ergebnisseite mit Präsentation der Synonyme als klickbare, alternative Suchbegriffe*

8 Praktische Erfahrungen und Ausblicke

Persönliche Gespräche mit einigen regelmäßigen Besuchern des Austria-Forums haben gezeigt, dass vor allem die Ausrichtung der Text-Transformation[1] auf die deutsche Sprache eine merkliche Qualitätssteigerung der Suche bewirkt hat. Die damit verbundenen Verbesserungen bei Suchbegriffen mit Umlauten stachen besonders ins Auge.

Bemerkenswert ist, dass auch Anwender die die Suche im Austria-Forum häufig verwenden, die Einstellungsmöglichkeiten der benutzerdefinierten Suche offenbar kaum nutzen. Im Grundtenor erscheinen die Optionen für Besucher des Forums zu komplex und die Wünsche tendieren zu einer möglichst einfachen Suche.

Anschließend sollen einige persönliche Eindrücke und Erkenntnisse vermittelt werden, die während der praktischen Umsetzung der vorgestellten Modifikationen an der Suche im Austria-Forum gesammelt wurden. Der Kritik werden auch Anmerkungen und Ideen bezüglich einer weiteren Verbesserung des Forums beziehungsweise der integrierten Such-Funktionalitäten beigefügt. Dadurch soll eine Diskussionsgrundlage für Weiterentwicklungen des Austria-Forums geboten und mögliches Potenzial aufgezeigt werden.

8.1 Lucene

8.1.1 Erfahrungen

leistungsfähig und intuitiv

Diese Bibliothek überzeugte in zweierlei Hinsicht: Einerseits stellte sich Lucene als umfangreiches, mächtiges und leistungsfähiges Werkzeug mit hohem Potenzial dar, andererseits ist die Handhabung aus Sicht eines Programmierers zeitgleich sehr intuitiv, schnell erlernbar und gut dokumentiert! Gerade diese Kombination ist wohl Grund für die hohe Popularität von Lucene.

1 siehe Seite 83

Zur Integration einer Suche ins Austria-Forum ist diese Bibliothek eine ausgezeichnete Wahl. Sie lässt sich praktisch nahtlos an die anderen Open-Source Java-Komponenten dieser Anwendung anknüpfen. Aus technologischer Sicht ergibt sich damit eine sehr stimmige Systemlandschaft.

8.1.2 Ausblicke

Mit der Abstimmung auf deutschsprachige Dokumente, den verschiedenen Optionen bei der benutzerdefinierten Suche und der Anbindung von GermaNet ist ein Bündel an Funktionalitäten gegeben, das sicherlich hohen Nutzen für die Anwender bringt. Dennoch seien einige Aspekte aufgezeigt, die eventuell Raum für weiterführende Verbesserungen bieten:

Metadaten-Extraktion: Ein Blick mit Luke[1] in den erstellten Index lässt vermuten, dass die Extraktion von Metadaten vor allem im Bereich der Biographien nicht einwandfrei funktioniert. Es scheint als ob Jahreszahlen aus den Metadaten irrtümlich als eigene Felder angelegt würden, wie in Abbildung 8.1 zu sehen ist. Eine genauere Untersuchung des Index könnte helfen, eventuelle Lücken und Missstände bei der Indexierung aufzudecken. Dadurch könnte der Inhalt des Index — und damit die Grundlage für jede Suche — qualitativ verbessert werden.

Beitrags-Typ: Das Austria-Forum beinhaltet sowohl redaktionell geprüfte Artikel (Stichwort: AEIOU), als auch nicht verifizierte Beiträge (Stichwort: Community). Bei der Präsentation der Suchergebnisse wäre eventuell die Information, um welchen Typus es sich beim jeweiligen Resultat handelt, wünschenswert.

Technisch könnte dies möglicherweise durch ein „Typ-Feld“ im Index zur Kennzeichnung des Beitrags-Typs beziehungsweise ein kleines Symbol bei der Ergebnispräsentation realisiert werden. Der Benutzer würde auf diese Weise darüber informiert, ob ein Ergebnis redaktionell geprüft und eventuell zitierbar ist, ohne den Eintrag dafür öffnen zu müssen. Somit wäre bei der Ergebnisseite neben der Relevanz auch ein Anhaltspunkt für die Qualität der Suchergebnisse gegeben. Dadurch soll auf keine Fall unterstellt werden, dass nicht verifizierte Beiträge von minderer Qualität seien! Vielmehr würden redaktionell geprüfte Einträge sozusagen mit einem Stempel *„Qualitätsgeprüft“* gekennzeichnet. Zusätzlich könnte der Beitrags-Typ bei der benutzerdefinierten Suche als Filterkriterium angeboten werden.

1 siehe Abschnitt 3.3 auf Seite 63

Status: Auch die Darstellung des Status des jeweiligen Ergebnisses — also die Information ob der betreffende Beitrag gesperrt oder editierbar ist — könnte womöglich von Interesse sein. Auch diese Erweiterung wäre mit einem zusätzlichen Feld im Index realisierbar.

Autor: Die Anführung des Autors des jeweiligen Resultats auf der Ergebnisseite wäre ebenfalls eine Information, die einen weiteren Gewinn für die Anwender darstellen könnte. Der Verfasser wird bereits im Feld „author“ indexiert.

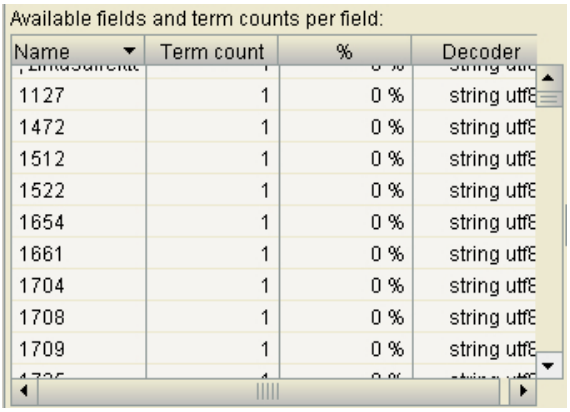

Abbildung 8.1: *Indexierung von Jahresangaben aus Metadaten als eigene Felder*

8.2 GermaNet

8.2.1 Erfahrungen

GermaNet stellt eine gute Möglichkeit dar, um Benutzern in deutschsprachigen Anwendungen bei einer Umformulierung der Suchanfrage Unterstützung anzubieten. In Kapitel 7 wurde darauf hingewiesen, dass bei der Suche im Austria-Forum GermaNet lediglich zur Synonymsuche verwendet wird. Es wurde hier also nur ein Teil des Potenzials dieses Wortnetzes ausgeschöpft. In anderen Applikationen können weitere abgebildete Relationen wie Meronymie und Hyponymie[1] ebenfalls von großem Nutzen sein.

Die abgebildeten Konzepte und Relationen sowie der Umfang des Wortnetzes erlauben es, eine Vielzahl verschiedener Aufgaben rund um deutsche Texte zu

1 siehe Seite 98

erfüllen. Im Austria-Forum stellt die implementierte Darbietung von synonymen Suchbegriffen eine Bereicherung für den Anwender dar. Hier fügt sich die intuitive Java Applikationsschnittstelle ebenso problemlos wie harmonisch in die Gesamtarchitektur ein.

8.2.2 Ausblicke

Potenzial: Im Rahmen einer Suchapplikation stellt das Anbieten von synonymen Suchbegriffen wohl die bestmögliche Verwendung eines angebundenen Wortnetzes dar. Wie bereits auf Seite 99 erwähnt, würde die Verwendung von anderen Relationen[1] die Bedeutung der Suchanfrage verändern. Aus dieser Sicht scheint das Potenzial von GermaNet im Kontext des Austria-Forums ausgereizt zu sein.

Anbindung: Begrüßenswert wäre jedoch ein alternativer Weg zur Einbindung ins Austria-Forum. Im Zusammenhang mit Lucene wäre eine Integration durch Indexierung des Wortnetzes — wie dies beispielsweise vom Princeton WordNet® per se unterstützt wird — ein gangbarer Weg um eine noch leistungsfähigere Nutzung zu erlauben.

Realisierbar wäre dies durch Indexierung der Synsets bei der Initialisierung des Austria-Forums, eventuell durch eine separate Methode mit eigenem Crawler. Die einzelnen Konzepte — sprich: Wörter — aus der GermaNet Datenbasis könnten in ihrer ursprünglichen Form[2] indexiert werden und die bedeutungsgleichen Begriffe in einem separaten Feld abgelegt werden. Während der Suche könnten auf diese Weise auch Synonyme für die eingegebenen Suchbegriffe direkt aus dem Index abgerufen werden. Die Java Applikationsschnittstelle von GermaNet wäre demnach nur in den Prozesses der Indexierung involviert und der Suchprozess könnte vollständig über Lucene und den Index abgewickelt werden. Somit ließe sich die Antwortzeit der Suche weiter verbessern und eine Vereinheitlichung der Abläufe bewerkstelligen.

1 wie Oberbegriff ↔ Unterbegriff, Teil ↔ Ganzes, siehe Seite 98
2 also ohne Text-Transformation, siehe *Feld-Option „Not Analyzed“* auf Seite 58

8.3 Austria-Forum Implementation

8.3.1 Erfahrungen

In diesem Forum wurden etliche verschiedenartige Anforderungen implementiert. Technologisch betrachtet fügen sich diese Komponenten nahtlos aneinander und ergeben in ihrer Gesamtheit eine homogene Anwendung.

Umsetzung

Im Detail lässt sich jedoch bald erkennen, dass im Laufe der Zeit viele verschiedene Entwickler mehr oder minder unabhängig voneinander einzelne Teile dazu beigetragen haben. Das äußert sich hie und da in unterschiedlichen Ansätzen für die selbe Problemstellung, einer Reihe von „Copy & Paste-Fragmenten“ und dem einen oder anderen Abschnitt mit obsoletem Code. Solche Fragmente erschweren nicht nur die Einarbeitung für weitere Entwickler, sie verkomplizieren auch allfällig nötige Änderungen und Erweiterungen. Ein ungehemmter Wildwuchs von Code birgt die Gefahr in sich, dass die Abläufe früher oder später undurchsichtig werden.

Dokumentation

Ein weiteres Manko stellt die praktisch nicht vorhandene Dokumentation abseits des Quellcodes dar. Eine bis zu einem gewissen Grad ausgereifte Dokumentation in bildlicher Form — wie etwa Architekturbilder, Klassen- und Ablaufdiagramme — würden vermutlich helfen eine klare Struktur in der Umsetzung zu bewahren. Auch der Einstieg für weitere Mitwirkende könnte auf diese Weise wesentlich erleichtert werden. Selbst wenn der Code verständlich und kommentiert ist, sagt ein Bild letztendlich doch oft mehr als tausend Worte. Ein gewisser Standard an Dokumentation außerhalb des Quellcodes wäre hier wünschenswert, damit die zugrunde liegenden Konzepte und Ideen leichter langfristig und durchgängig erhalten bleiben.

8.3.2 Ausblicke

Auch im Bereich der Implementation und Funktionalität gibt es noch Raum für Verbesserungen und Erweiterungen.

Umsetzung und Dokumentation: Beide zuvor genannten Kritikpunkte könnten beispielsweise im Rahmen eines Projekts gleichzeitig behoben werden. Der Aufwand für eine Überarbeitung des Codes und gleichzeitige Erstellung der Dokumentation durch ein Projektteam sollte sich in Grenzen halten lassen können.

Ein möglicher Ansatz wäre eine Überarbeitung nach einem Konzept ähnlich dem *Extreme Programming XP*. Die einzelnen bereits implementierten

Funktionen im Austria-Forum stellen die Anwendungsfälle dar. Die Iterationen ließen sich nach Themenbereichen wie „Suche“, „Benutzerverwaltung“, „Beitragserstellung“ etc. gliedern. Anstelle der Implementation würden hier nur mehr eine Überarbeitung des Codes und Erstellung der Dokumentation von Nöten sein. Durch die für XP typischen Ansätze wie *paarweises Arbeiten* und *Wechsel der Teams* lässt sich das Verständnis der Architektur leichter im Projektteam verbreiten. Somit wäre es möglich die Konzepte rasch auf mehrere Know How Träger zu verteilen. Ein einheitlicheres Bild der gesamten Anwendung im Kreise der Entwickler und eine gemeinsame Handschrift bei der Umsetzung würden die Wartbarkeit des Austria-Forums sicherlich erleichtern und die Flexibilität für künftige Erweiterungen wahren.

Metadaten — Schlüsselwörter: Auf Seite 108 wurde bereits erwähnt, dass die Inspektion des Indexes einen Handlungsbedarf im Bereich der eingebbaren Metadaten — den Schlüsselworten — aufzeigt. Ausmerzen dieses Problems und eine vermehrte Verwendung dieser Möglichkeit zur Eingabe von expliziten Schlüsselwörtern könnten die Suchergebnisse durchaus positiv beeinflussen.

Auto-Vervollständigung: Die Bedienerfreundlichkeit bei der Eingabe der Suchanfrage könnte durch eine Auto-Vervollständigung anhand der indexierten Terme weiter verbessert werden. Tippfehler in den Suchbegriffen könnten auf diese Weise ebenfalls minimiert werden. Erfolgreiche Suchmaschinen wie Google legen hier die Latte sehr hoch.

Die angeführten Eindrücke und Denkanstöße haben auf einige Aspekte rund um das Thema *„Suche im Austria-Forum“* aufgezeigt. Abschließend kann das folgende Fazit aus dieser Arbeit gezogen werden.

Konklusion

Suchmaschinen stellen sich als komplexe Thematik dar, in deren Bereich eine Fülle von unterschiedlichen Anforderungen und Konzepten fallen. Das stetige Wachstum der zugrunde liegenden Theorie versucht mit den rasant steigenden Ansprüchen und Einsatzgebieten Schritt zu halten. Dieses Feld bietet mannigfaltige Möglichkeiten zur weiteren Entfaltung und ein Ende der Entwicklung ist nicht abzusehen. Suchmaschinen sind längst über das Internet hinausgewachsen und erschließen immer weitere Bereiche in der Informationstechnologie.

Im praktischen Anwendungsfall des Austria-Forums sind viele Ansätze moderner Suchmaschinen umgesetzt. Die verwendete Java-Bibliothek Lucene integriert sich dabei sehr gut in die Architektur der JSPWiki Anwendung. Im Zuge des praktischen Teils dieser Masterarbeit konnten einige Verbesserungen und Erweiterungen realisiert werden, die für die Benutzer des Austria-Forums von Nutzen sind. Dabei sticht die Anpassung der Text-Transformation auf deutschsprachige Quelldokumente besonders hervor. Die gewonnenen Erkenntnisse rücken Lucene dabei in ein positives Licht.

Auch die Anbindung von GermaNet zur Synonymsuche erweitert die Such-Funktionalität im Austria-Forum um eine Annehmlichkeit für die Anwender. Die Java Applikationsschnittstelle fügt sich dabei harmonisch in die umgebende Anwendung ein und ermöglicht problemlos den Zugriff auf das Wortnetz.

Die *„Suche im Austria-Forum“* erweist sich als spannendes Umfeld für eine Auseinandersetzung mit dem Thema „Suche“ an sich. Der Grund liegt in der Vielzahl der angewandten Konzepte wie Ähnlichkeitssuche, Synonymsuche und dergleichen. Die praktischen Erfahrungen zeigen, dass auch in diesem Kontext noch immer Entwicklungspotenzial im Austria-Forum gegeben ist.

Allgemein kann gesagt werden, dass Suchfunktionalitäten mittlerweile in vielen Anwendungen zu unverzichtbaren Navigationshilfen und wesentlichen Bestandteilen dieser Applikationen herangereift sind. Benutzer haben zunehmend das Bedürfnis gleichermaßen intuitive wie effektive Suchwerkzeuge zur Hand zu haben. Aus diesem Grund verdienen Suchmaschinen ein ausreichendes Maß an Aufmerksamkeit.

Literaturverzeichnis

[ACKC10] Ao-Jan Su ; Charlie Hu ; Kuzmanovic, Aleksandar ; Cheng-Kok Koh: How to Improve Your Google Ranking: Myths and Reality. In: *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on* Bd. 1, 2010, S. 50 –57

[ACR07] Amati, Giambattista (Hrsg.) ; Carpineto, Claudio (Hrsg.) ; Romano, Giovanni (Hrsg.): *Advances in information retrieval: 32th European Conference on IR Research*. Springer-Verlag Berlin Heidelberg, 2007. – ISBN 978–540–71494–1

[AEI] *AEIOU - Das Annotierbare Elektronische Interaktive Oesterreichische Universal-Informattionssystem*. Online. http://www.aeiou.at/, Abruf: 18.5.2012

[AFI10] *Das Austria-Forum – die Internet-Wissenssammlung für Österreich*. Online. http://www.austria-lexikon.at/attach/Infos_zum_AF/Pr%C3%A4sentation_Austria-Forum/Austria-Forum_Information.pdf. Version: 2010, Abruf: 20.5.2012

[And11] Andrews, Keith: *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Online. http://ftp.iicm.tugraz.at/pub/keith/thesis/. Version: 2011, Abruf: 3.4.2012

[Aus] Austria-Forum Community: *Austria-Forum, Das Österreichische Wissensnetz*. Online. http://www.austria-lexikon.at/af/, Abruf: 27.12.2011

[BDH03] Barroso, Luiz A. ; Dean, Jeffrey ; Hölzle, Urs: *Web search for a planet: The Google cluster architecture*. Online. http://research.google.com/archive/googlecluster.html. Version: 2003, Abruf: 04.02.2012

[BGH+06] Bestehorn, Markus ; Gitzinger, Thomas ; Heitmann, Benjamin ; Kappler, Thomas ; Krestel, Ralf ; Lang, Tobias ; Leitner, Johannes ; Siegmund, Carsten ; Wild, Florian: Text Mining: Wissensgewinnung aus natürlichsprachigen Dokumenten / Institut für Programmstrukturen und Datenorganisation (IPD) der Fakultät für

Informatik an der Universität Karlsruhe (TH). Dr. René Witte und Jutta Mülle, 2006 (5). – Forschungsbericht. – Hauptseminar im Wintersemester 2004/2005

[Bia11] Bialecki, Andrzej: *Luke - Lucene Index Toolbox*. Online. http://code.google.com/p/luke/. Version: 2011, Abruf: 14.3.2012

[BR08] Butler, Mark H. ; Rutherford, James: Distributed Lucene : A distributed free text index for Hadoop / HP Laboratories. 2008 (64). – Forschungsbericht

[BS09] Becker, Konrad ; Stalder, Felix: *Deep Search: Politik des Suchens jenseits von Google*. StudienVerlag Ges.m.b.H, 2009. – ISBN 978–3–7065–4794–9

[Bus45] Bush, Vannevar: As We May Think. In: *The Atlantic Monthly* 176 (1945), S. 101–108

[BYRN11] Baeza-Yates, Ricardo ; Ribeiro-Neto, Berthier: *Modern Information Retrieval: The concepts and technology behind search*. 2. Addison-Wesley, 2011. – ISBN 978–0321416919

[CMS10] Croft, W. B. ; Metzler, Donald ; Strohman, Trevor: *Search Engines: Information retrieval in practice*. Pearson Education Inc., 2010. – ISBN 978–0–13–136489–9

[Col05] Cole, B.: Search engines tackle the desktop. In: *Computer* 38 (2005), März, Nr. 3, S. 14 – 17. http://dx.doi.org/10.1109/MC.2005.103. – DOI 10.1109/MC.2005.103. – ISSN 0018–9162

[Cut05] Cutting, Doug: *Open Source Search*. Online. http://www.research.ibm.com/haifa/Workshops/ir2005/papers/DougCutting-Haifa05.pdf. Version: 2005, Abruf: 3-3-2012

[Deu00] Deutsch, Peter: Archie - a Darwinian development process. In: *Internet Computing, IEEE* 4 (2000), S. 69–71

[DG08] Dean, Jeffrey ; Ghemawat, Sanjay: MapReduce : Simplified Data Processing on Large Clusters. In: *Communications of the ACM* 51 (2008), S. 1–13

[DG11] Doran, Derek ; Gokhale, Swapna: Web robot detection techniques: overview and limitations. In: *Data Mining and Knowledge Discovery* 22 (2011), S. 183–210. – ISSN 1384–5810. – 10.1007/s10618-010-0180-z

[Die11] Diem, Peter: *Das Austria-Forum - Die österreichische Internet-Enzyklopädie*. Online. http://www.austria-lexikon.at/attach/Infos_zum_AF/11-12-10-Austria-Forum.pdf. Version: 2011, Abruf: 20.5.2012

[EE10] Edosomwan, Joseph ; Edosomwan, Taiwo O.: Comparative analysis of some search engines. In: *South African Journal of Science* 106(11/12) (2010), Nr. 169, S. 4

[Ger09] *GermaNet Homepage*. Online. http://www.sfs.uni-tuebingen.de/lsd/. Version: 2009, Abruf: 14.3.2012

[Gol08] Golliher, Sean A.: Search Engine Ranking Variables and Algorithms. In: *SEMJ.ORG* 1 (2008), S. 15–19

[Hel08] Helic, Denis: *Multimedia Information Systems: Web Search Engines*. Online. http://coronet.iicm.edu/lectures/mmis/material/slides_search.html. Version: 2008, Abruf: 23.7.2012

[Hen08] Henrich, Andreas: *Information Retrieval 1: Grundlagen, Modelle und Anwendungen*. Online. http://www.uni-bamberg.de/minf/IR1-Buch. Version: 2008, Abruf: 28.2.2012

[HMW08] Helic, Denis ; Maurer, Hermann ; White, Bebo: Austria-Forum: A citable web encyclopedia. In: *IADIS International Conference WWW/Internet*, 2008, S. 19–26

[Joa02] Joachims, Thorsten: Optimizing search engines using clickthrough data. In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA : ACM, 2002 (KDD '02). – ISBN 1–58113–567–X, S. 133–142

[Jsp09] *JSPWiki Homepage*. Online. http://www.jspwiki.org/. Version: 2009, Abruf: 20.5.2012

[Kon08] Konchady, Manu: *Building Search Applications: Lucene, LingPipe, and Gate*. 1. Mustru Publishing, 2008. – ISBN 978–0–61520–425–3

[Lem11] Lemur Project Contributors: *The Lemur Project*. Online. http://lemurproject.org/. Version: 2011, Abruf: 18.3.2012

[Len11] Lenič, Mitja: *Lucenetransform: Transparent transformation (compression) of lucene index directory*. Online. http://code.google.com/p/lucenetransform/. Version: 2011, Abruf: 13.3.2012

[LK07] Lemnitzer, Lothar ; Kunze, Claudia: *Computerlexikographie: Eine Einführung*. Gunter Narr Verlag, 2007. – 351 S. – ISBN 978–3–8233–6315–6

[MD10] Maurer, Hermann ; Diem, Peter: *Die Internet-Enzyklopädie 'Austria-Forum'*. Online. http://austria-lexikon.at/af/Forschung. Version: 2010, Abruf: 20.5.2012

[MHG10] McCandless, Michael ; Hatcher, Eric ; Gospodnetić, Otis: *Lucene in Action*. 2. Manning Publications Co., 2010. – ISBN 978–1933988177

[MM12] Maurer, Hermann ; Müller, Heimo: *Das Austria-Forum und die Informationswissenschaft*. Online. http://www.austria-lexikon.at/attach/Infos_zum_AF/Gesamtdarstellung_des_AF.pdf. Version: 2012, Abruf: 20.5.2012

[MMG+11] Mukhopadhyay, Debajyoti ; Mukherjee, Sajal ; Ghosh, Soumya ; Kar, Saheli ; Kim, Young-Chon: Architecture of A Scalable Dynamic Parallel WebCrawler with High Speed Downloadable Capability for a Web Search Engine. In: *CoRR* abs/1102.0676 (2011), S. 103–108

[MRS08] Manning, Christopher D. ; Raghvan, Prabhakar ; Schütze, Hinrich: *Introduction to Information Retrieval*. Cambridge University Press, 2008. – ISBN 0521865719. – Online edition http://www-nlp.stanford.edu/IR-book/

[PB98] Page, Larry ; Brin, Sergey: The anatomy of a large-scale hypertextual Web search engine. In: *Computer Networks and ISDN Systems* 30 (1998), S. 107–117

[Rei10] Reischl, Gerald: *Die Google Falle: Die unkontrollierte Weltmacht im Internet*. 7. Verlag Carl Ueberreuter, Wien, 2010. – ISBN 978–3–8000–7323–8

[SCG10] Sun, Yang ; Council, Isaac G. ; Giles, C.Lee: The Ethicality of Web Crawlers. In: *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on* Bd. 1, 2010, S. 668 –675

[SEW07] SEW Staff: *How Search Engines Rank Web Pages*. Online. http://searchenginewatch.com/article/2064539/How-Search-Engines-Rank-Web-Pages. Version: 2007, Abruf: 25.2.2012

[Sin01] Singhal, Amit: Modern Information Retrieval: A Brief Overview. In: *IEEE Data(base) Engineering Bulletin* 24 (2001), S. 35–43

[Sko10] Skorkin, Alan: *How Search Engines Process Documents Before Indexing*. Online. http://www.skorks.com/2010/03/how-search-engines-process-documents-before-indexing/. Version: 2010, Abruf: 17.2.2012

[Sla10] Slawsky, Bill: *What Makes a Good Seed Site for Search Engine Web Crawls?* Online. http://www.seobythesea.com/2010/05/what-makes-a-good-seed-site-for-search-engine-web-crawls/. Version: 2010, Abruf: 12.2.2012

[SLP11] Shin, Yongwook ; Lim, Junseok ; Park, Jonghun: Joint Optimization of Index Freshness and Coverage in Real-Time Search Engines. In: *Knowledge and Data Engineering, IEEE Transactions on* PP (2011), Nr. 99, S. 1. http://dx.doi.org/10.1109/TKDE.2011.144. – DOI 10.1109/TKDE.2011.144. – ISSN 1041–4347

[Sno10] *Tartarus Snowball Homepage*. Online. http://snowball.tartarus.org/. Version: 2010, Abruf: 30.3.2012

[Sun] Sun Sirius GmbH: *Suchmaschinen Verzeichnis und Informationen*. Online. http://www.suchmaschinen-online.de/technik/geschichte.htm, Abruf: 20.7.2012

[The06] The Apache Software Foundation: *Lucene Java Documentation*. Online. http://lucene.apache.org/core/old_versioned_docs//versions/3_0_2/index.html. Version: 2006, Abruf: 12.3.2012

[Tra09] Trattner, Christoph: *Vom Austria-Forum zum Wiki-Konzept*, Technische Universität Graz, Master thesis, 2009

[Wal] Wall, Aaron: *Search Engine History*. Online. http://www.searchenginehistory.com/, Abruf: 20.7.2012

[Wol12] Wolfram Alpha LLC—A Wolfram Research Company: *About Wolfram|Alpha - Making the world's knowledge computable*. Online. http://www.wolframalpha.com/about.html. Version: 2012, Abruf: 3.3.2012

[YDS09] Yan, Hao ; Ding, Shuai ; Suel, Torsten: Inverted Index Compression and Query Processing with Optimized Document Ordering. In: *Proceedings of the 18th international conference on World wide web*, 2009, S. 401–410

[YL96] Yuwono, Budi ; Lee, Dik L.: Search and ranking algorithms for locating resources on the World Wide Web. In: *Data Engineering, 1996. Proceedings of the Twelfth International Conference on*, 1996, S. 164 –171

[YMH02] Yuan, X. ; MacGregor, M.H. ; Harms, J.: An efficient scheme to remove crawler traffic from the Internet. In: *Computer Communications and Networks, 2002. Proceedings. Eleventh International Conference on*, 2002. – ISSN 1095–2055, S. 90–95

Abbildungsverzeichnis

Tabellenverzeichnis

Stichwortverzeichnis