Markus Heller

Fachspezifische Indexierung von historischen Dokumenten II
Ein Framework zur approximativen Indexierung semistrukturierter Dokumente

aus:

Forschung in der digitalen Welt

Sicherung, Erschließung und Aufbereitung von Wissensbeständen

Herausgegeben von Rainer Hering, Jürgen Sarnowsky, Christoph Schäfer und Udo Schäfer

S. 59–84

Impressum

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.
Die Online-Version dieser Publikation ist auf der Verlagswebsite frei verfügbar *(open access)*. Die Deutsche Nationalbibliothek hat die Netzpublikation archiviert. Diese ist dauerhaft auf dem Archivserver der Deutschen Nationalbibliothek verfügbar.

Open access über die folgenden Webseiten:
Hamburg University Press – http://hup.sub.uni-hamburg.de
Archivserver der Deutschen Nationalbibliothek – http://deposit.d-nb.de

ISBN-10 3-937816-27-5 (Printausgabe)
ISBN-13 978-3-937816-27-2 (Printausgabe)
ISSN 0436-6638 (Printausgabe)

Produktion: Elbe-Werkstätten GmbH, Hamburg, Deutschland
http://www.ew-gmbh.de

Bildnachweis: Der Abdruck aller Abbildungen erfolgt mit freundlicher Genehmigung der Autoren bzw. des Autors des jeweiligen Beitrags.

Inhaltsübersicht

Fachspezifische Indexierung von historischen Dokumenten II

Ein Framework zur approximativen Indexierung semistrukturierter Dokumente

Markus Heller

1. Einleitung

Die Nutzung von Rechnern in den Geschichtswissenschaften ist inzwischen weit verbreitet: Niemand schreibt seine Aufsätze mehr mit der Schreibmaschine und jeder speichert seine Entwürfe digital auf Datenträgern. Auch Quellen werden inzwischen mit Computern bearbeitet: Man bedient sich generischer Programme aus den Office-Schienen der großen Hersteller und baut auf diese Weise seine Corpora auf.

Solange derartige Editionsansätze auf ein lokales Nutzerumfeld beschränkt bleiben, mag der fehlende methodische Ansatz im Umgang mit diesen Ressourcen den geringen Einsatz rechtfertigen. Auf diese Weise ist jedoch in den vergangenen Jahren eine bunte Landschaft von hinsichtlich der Datenformate inkompatiblen und höchst heterogenen Datenbeständen gewachsen, wobei die Nutzbarkeit der Daten durch zukünftige Historikergenerationen alles andere als gesichert ist. Sind heute schon die älteren Winword-Dateiformate mit aktueller Software vom gleichen Hersteller nicht mehr lesbar, so tritt dieses Problem bei unterschiedlichen Herstellern mit proprietären Datenformaten noch in viel schärferer Form auf. Das oftmals fehlende Bewusstsein, auf offene Speicherformate und Standards zu setzen, gleicht einer fahrlässigen Strategie, öffentliches Geld und die wertvolle Lebenszeit vieler Quelleneditoren gleichsam zu vergeuden. Stattdessen liegt es nahe, wenn Editionsprojekte bei der Entscheidung über die Fi-

nanzierung maßgeblich dahingehend beurteilt werden, wie sehr sie sich der Forderung beugen, dass sie unabhängig von ihrer geographischen Speicherung und unabhängig vom zukünftigen Bearbeitungswerkzeug des Benutzers durchsucht werden können.

Dieses Szenario verlangt nach einer verteilten Sucharchitektur, in der die Daten in einem offen gelegten Format vorgehalten, zentral indexiert und mit einer ebenfalls zentralen Schnittstelle durchsuchbar gemacht werden. Dabei wird auch deutlich, dass die allgemein öffentlich verfügbare Suchtechnik von Google, Yahoo und Co. deutliche Schwächen aufweist: Sie sind auf verlinkte Webseiten unstrukturierten Inhalts spezialisiert und nehmen keinerlei Rücksicht auf Dokumente mit Auszeichnungen und hinterlegten fachlichen Zusatzinformationen. Die Merkmale solcher semistrukturierter Dokumente können bislang bei der Suche nicht angegeben werden. Auch in Bezug auf den Inhalt der Dokumente zeigen sich Probleme: Falls ein Name oder eine Bezeichnung in einem Dokument auch nur geringfügig abweichend von der Suchanfrage geschrieben wird, besteht mit der gegenwärtig öffentlich verfügbaren Suchtechnik keine Möglichkeit, diese Stellen zu finden.

Es wird also deutlich, dass ein dringender Bedarf zur Entwicklung von Suchmaschinentechnologien besteht, um die Arbeit der Editoren von heute nicht wertlos erscheinen zu lassen.

2. Problemstellung

Die zu beschreibenden Probleme beginnen bei der Erfassung der Texte: Als Speicherformat bieten sich verschiedene XML-Standards an, wie etwa bei mittelalterlichen Urkunden der Standard CEI.[1] XML-Texte weisen jedoch selten eine Verlinkungsstruktur ähnlich der von HTML-Texten auf, obgleich dies grundsätzlich möglich wäre: XHTML ist eine völlig XML-konforme HTML-Variante. Ein Crawler, der alle Dokumente in einem im Inter-

[1] Adresse: http://www.cei.lmu.de/ (letzte Einsichtnahme am 17.05.2006). Zudem: Georg Vogeler: Europäisches Urkundenerbe. Zu Potentialen und Perspektiven eines internationalen Fachinformationssystems digitaler Urkundenpublikationen. In: Elektronische Fachinformationssysteme in der Geschichte. Jahrestagung der AGE, 25.–26.11. 2005. Hg. von Franz Götz. München 2006 (im Druck).

net verteilten Corpus erschießt, kann nur jene Dokumente finden, die von anderen Dokumenten aus referenziert und verlinkt sind.

Hinsichtlich der Techniken des Information Retrieval muss zum Verständnis der Technologie erklärt werden, dass eine Suchmaschine nicht erst zum Zeitpunkt der Anfrageverarbeitung beginnt, Dokumente zu suchen. Vielmehr gleicht eine Suchmaschine einem System, das einen Index eines Buches erstellt. Der Index wird lange vor dem Suchzugriff erstellt und zum Zeitpunkt der Anfragebearbeitung nur abgefragt. Aus diesem Grund unterscheidet sich jede Suchmaschinentechnologie konzeptionell von den klassischen Suchverfahren. Auch reguläre Ausdrücke dienen in erster Linie einer linearen Suche, und nicht einem Look-up in einem Index. Aus diesem Grund konzentriert sich unser Projekt nicht auf die Entwicklung von Suchverfahren, sondern auf Verfahren, in einem Index auf effiziente Weise die entsprechenden Einträge zu finden.

Bei flachen Texten gleicht nun die Dokumentenverarbeitung einer Suchmaschine der Erstellung eines Buchindizes. Doch XML-Texte weisen ein weiteres zentrales Merkmal auf, in dem sie sich von flachen Texten unterscheiden: Sie enthalten eine Struktur, anhand derer die verschiedenen Textblöcke ausgezeichnet und beschrieben werden. Mit Blick auf historische Quellentexte können mit derartigen Textstrukturen einerseits tatsächliche Eigenschaften und Gliederungsmerkmale des Originals, aber auch Hinweise eines Editors, Übersetzungen, Referenzen auf die Sekundärliteratur sowie auf andere Dokumente enthalten sein. Es liegt nun nahe, dass eine Suchmaschine zwischen all diesen verschiedenen Texttypen unterscheiden können sollte. Wer ausschließlich nach Originaltexten sucht, sollte ausschließlich Stellen in diesem Texttyp angeboten bekommen.

Interessant ist in diesem Kontext auch, dass Editoren zwar an die Einhaltung ihrer spezifischen XML-Syntaxdefinition (‚Schema', oder ‚Document Type Definition', bzw. DTD) gebunden sind, diese aber durchaus enorme Freiheitsgrade definieren kann. So kann ein XML-Schema unterschiedliche Auszeichnungstiefen zulassen. Es sind auch Rekursionen an Auszeichnungsmerkmalen möglich. Ebenfalls genannt werden muss an dieser Stelle die Tatsache, dass ein Ersteller eines Schemas die Semantik der Tags frei wählen kann. Wenn etwa ein Tag ‚document' definiert werden sollte, und damit gemeint ist, dass in jedem Tag ein einzelnes Quellendokument abgelegt werden darf, dann tritt die Situation ein, dass eine solche XML-Datei mehrere Quellendokumente enthält. Die Identität zwischen

‚Datei' und ‚Dokument' ist definitiv nicht mehr zwingend gegeben. Dies bedeutet, dass eine Suchmaschine für XML-Texte nicht Dateien finden will, sondern Stellen in einem XML-Corpus.

Im Allgemeinen wird von den Auszeichnungselementen in XML-Dateien von ‚Struktur' und von den Textinhalten von ‚Content' gesprochen. Wir haben nun bisher die Struktur betrachtet. Doch auch der Content weist Eigenschaften auf, welche klassische Suchmaschinen an den Rand ihrer Fähigkeiten führen. Betrachten wir die *Lebensgeschichte des Till Eulenspiegel*. Es handelt sich dabei um ein frühneuhochdeutsches Textbeispiel, das zu dieser Zeit weite Verbreitung fand und auch heute noch allgemein bekannt ist. Die elektronische Fassung eines Drucks von 1540[2] enthält dabei folgende Schreibungen desselben Namens:

- Ulenspiegel 288
- Ulenspiegels 6
- Ulenspigel 2
- Ulenspiegeln 1
- Ulenspieg 1
- Ulnspiegel 708
- Ulnspiegels 11
- Ulnspigel 1
- Ulnspiegln 1
- vlnspiegel 1
- Ulnspiegls 1
- Ulnspeigel 1
- Ulnspiegeln 1
- Ulnspiegl 1

Flexionen werden in dieser Aufzählung unterschieden, weil wir in erster Linie von einer Suchmaschine ausgehen, die keinerlei approximative Eigenschaften hat. Anhand dieses Beispiels wird die Eigenschaft von alten Texten deutlich, dass eine Orientierung an scheinbar verbindlichen Standards nicht angenommen werden kann.[3] Vielmehr ist zu beobachten, dass

[2] Adresse: http://www.eulenspiegel.is.guad.de/ (letzte Einsichtnahme am 17.05.2006).

[3] Ob dies angesichts der häufig revidierten Vorgaben der neuen deutschen Rechtschreibung angenommen werden kann, wäre eine zu weit führende Diskussion. Das Problem selbst ist doch im Grundsatz noch präsent. Ein Ansatz für einen Indexer für eine orthographisch nicht standardisierte Sprache findet sich hier: Jan Strunk: Information retrieval for languages that lack a fixed

eine Normierung von Schreibweisen erst mit der breiten Verfügbarkeit der Wörterbücher der Gebrüder Grimm (begonnen in der ersten Hälfte des 19. Jahrhunderts) und Johann Christoph Adelung (Ende 18. Jahrhundert) festgestellt werden kann. Die Eigenschaft der Informatik, grundsätzlich von ‚sauberen' und normierten Daten auszugehen, führt hier nicht weiter. Auch Google unterliegt diesem Phänomen: Bei der Eingabe von ‚Schiffahrt' und ‚Schifffahrt' erhält man unterschiedliche Treffermengen. Mit Blick auf den Content von in XML-Dateien erfassten Quellentexten sind jedoch Ansätze vonnöten, die in der Lage sind, abweichende Schreibungen ebenfalls zu finden. Diese Forderung entspricht dem Postulat der deskriptiven Linguistik, die versucht, die Sprachstände abzubilden und so zu akzeptieren, wie sie in den Quellen vorgefunden werden. Eine zu Recht stark kritisierte präskriptive Linguistik würde versuchen, den Quellentexten ex post eine Norm überzustülpen, die der diachronen, aber auch der synchron-dialektologischen Variabilität der deutschen Sprache nicht gerecht wird.

Zuletzt muss sich inzwischen jede Suchmaschine an den Interessen des Benutzers orientieren: Google versucht, über Gmail Interessenprofile der Nutzer zu gewinnen, und auf diesem Weg ein Ranking anzubieten, das den Neigungen des Gmail-Benutzers entgegenkommt. Die anonyme Suchschnittstelle ist dagegen nicht personalisiert, wo gleich bei einigen Herstellern dennoch über Cookies versucht wird, mehr über die Interessenlagen der Benutzer zu erfahren.

Eine Suchmaschine für historische Texte hat im Wesentlichen drei Nutzergruppen: Historiker, Diplomatiker und Linguisten. Jede der Benutzergruppen hat ihr eigenes Interessenprofil:

– Der Historiker interessiert sich in erster Linie für Inhalte von Urkunden, aber auch für Interpretationen und Einordnungen, die von einem Editor vorgenommen wurden. Weniger interessant sind Informationen über die äußere Form von Dokumenten. Eine approximative Suche ist schon mit Blick auf verschiedene Schreibweisen von Namen unabdingbar.
– Der Diplomatiker interessiert sich weniger für den Inhalt, dafür aber für Informationen zur Erstellung und zur äußeren Form. Da modernsprachliche Anmerkungen sich eher einer orthographischen Norm unterwerfen als alte Texte, ist ein approximativer Zugriff nicht von dersel-

orthography. 2003. Adresse: http://www.linguistics.ruhr-uni-bochum.de/~strunk/LSreport.pdf (letzte Einsichtnahme: 21.4.2006).

ben herausragenden Bedeutung wie bei Originaltexten. Dennoch ist eine Approximation in vielen Fällen hilfreich.
– Der Linguist interessiert sich bei einem synchronen Erkenntnisziel für verschiedene Schreibungen und Aussprachen in Bezug auf eine regionale Verteilung. Bei diachronen Fragestellungen ist von Interesse, wie sich bestimmte Begriffe unter Einschränkung auf bestimmte Regionen in ihrer Form verändert haben. Dabei ist denkbar, dass einerseits approximative Fragestellungen, aber auch die Suche nach bestimmten Einzelformen von Interesse sein können. Weniger von Interesse ist für Linguisten die Suche in Annotationen.

Auf jeden Fall wird deutlich, dass bei einer Suchmaschine für semistrukturierte alte Texte der Einsatz von Approximation vom Benutzer steuerbar sein muss. Hinsichtlich der Gewichtung von Feldern sollte der Benutzer eine Möglichkeit geboten bekommen, ein seinen Interessen entsprechendes Profil auszuwählen. Die Approximation erfolgt dabei stets bezogen auf den Content. Der Strukturanteil der Suchanfrage wird dabei durch den Suchenden entweder fest vorgegeben oder in Teilen völlig freigestellt.
Neben einer direkten, einstufigen Suche besteht ein weiteres Suchschema in dem Ansatz, dass ein Benutzer nicht explizit nach einem bestimmten Begriff sucht, sondern dass er einen Sammelbegriff eingibt, um in einem zweiten Schritt seine Suche einzugrenzen. Die Funktion, die diese Möglichkeit zur Verfügung stellt, wird ‚Dynamic Drilldown' genannt und wird von ‚FAST Data Search'[4], aber auch von der Suchimplementierung von Exalead[5] geboten. Sie setzt voraus, dass Extraktoren definiert wurden, die bestimmte Dokumenteneigenschaften oder Entitäten mit einer bestimmten Eigenschaft auszeichnen. Wie bei Exalead sind ‚Drilldowns' auf Eigennamen, Ortsnamen, Regionen, Sprachen, Dokumententypen, Archive usw. möglich. Ein derartiger Zugriff würde der Fragestellung entgegenkommen, nach der ein Historiker etwa mehr über das geographische oder personelle Umfeld einer Person oder eines Vorgangs wissen will.

Das bislang bezeichnete Szenario ist sehr umfangreich und im Rahmen eines kleinen Projekts nur äußerst bedingt umsetzbar. Mitunter aus diesem Grund haben wir uns entschlossen, nur die wesentlichen Technologiefragen zu klären und Themenbereiche auszuklammern, von denen wir über-

[4] Adresse: http://www.fastsearch.no (letzte Einsichtnahme am 17.05.2006).

[5] Adresse: http://www.exalead.com (letzte Einsichtnahme am 17.05.2006).

zeugt sind, dass sie weit über den fachlichen Horizont der Computerlinguistik bzw. Historischen Fachinformatik hinausführen. Darunter fällt beispielsweise die gesamte Fragestellung um die Erschließung verteilter Corpora: Wir gehen davon aus, dass die Texte bereits vorliegen. Ein weiterer Bereich, der besonders von Usability-Spezialisten gelöst werden muss, ist jener des Benutzerfrontends. Da die Firma Arpa Data GmbH[6] Interesse an einer Softwarebibliothek hat, die in die bestehende OCR-Lösung integriert werden kann, wollen wir uns darauf beschränken, ein objektorientiertes Framework zu entwickeln, mit dem die bezeichneten Fragestellungen des Information Retrieval grundsätzlich und für unsere Belange exemplarisch gelöst werden können.

3. Projektbeschreibung

Das beschriebene Projekt hat sich zum Ziel gesetzt, eine Indexierungsarchitektur zu entwickeln, die gemischte, also Content- und Strukturanfragen (‚CAS-Queries') beantworten kann, wobei der Content-Anteil über Approximationsverfahren im Zugriff auf den Index aufgeweicht werden soll. Auf diese Weise können Anfragen bearbeitet werden, deren Strukturanteil sich auf die Auszeichnung durch einen Fachmann bezieht, und deren Content-Anteil mit möglichen abweichenden Schreibungen in Deckung gebracht werden kann.

Es gibt bereits seit einigen Jahren Datenbanksysteme, die nativ mit XML-Texten umgehen und CAS-Abfragen beantworten können.[7] Allerdings ist zu beobachten, dass bis dato keine XML-Datenbanken oder native XML-Indizes bekannt sind, die eine Approximation auf den Content- oder Strukturanteil implementieren. Unsere Entscheidung, einen eigenen Index zu entwickeln, lag darin begründet, dass der Aufwand, eine etablierte Matching-Engine[8] einer Datenbank auf approximative Fähigkeiten hin zu erweitern, im Arbeitsumfang schwer kalkulierbar ist. Es besteht das nicht un-

[6] Adresse: http://www.arpa.ch (letzte Einsichtnahme am 17.05.2006). Arpa Data beschäftigt sich vor allem mit der OCR-technischen Erfassung von Zeitungsarchiven.

[7] Vgl. Harald Schöning: Tamino – A Database System Combining Text Retrieval and XML. In: Intelligent Search on XML Data. Applications, languages, models, implementations, and benchmarks. Ed. by Henk Blanken. Berlin 2003. S. 77–94. Auch: Cynthia M. Saracco: Query DB2 XML Data with SQL 16.03.2006. Adresse: http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0603saracco2/ (letzte Einsichtnahme 06.04.2006).

erhebliche Risiko, dass ein neues Release aus dem Hause des Herstellers die hinzugefügten Approximationsfunktionen nicht unterstützt oder die Modifikationen inkompatibel sind. Aus diesem Grund haben wir uns zu einer Eigenentwicklung entschlossen, zumal ein großer Teil der benötigten Teilalgorithmen an unserem Institut oder in Kooperation entwickelt wurden. Durch die Möglichkeit, die Rahmenbedingungen der Architektur selbst zu definieren, konnten wir auch die Anforderungen aus dem Retrieval in historischen Textcorpora einfließen lassen.

In Bezug auf historische Texte ist erkennbar, dass die Contentapproximation nicht nur alphabetisch, sondern auch gesondert numerisch erfolgen muss, wenn etwa nach Dokumenten um das Jahr 1400 gefragt wird: Es ist neben dem alphabetischen ein weiterer Lookup für Zahlen und Jahresangaben erforderlich. Über eine konfigurationsgesteuerte Anwahl eines bestimmten Indextyps werden als Zahlen erkannte bzw. in Zahlen konvertierte Entitäten typenorientiert geroutet. Die Unterscheidung ist aus dem Grund wichtig, weil sich bei einer alphabetischen Approximation die Fehlerschwere zwischen den verschiedenen Stellen eines Wortes kaum unterscheidet. Bei Zahlen ist die Fehlerschwere aber sehr wohl an die Stelle gebunden, denn der Unterschied zwischen „1991“ und „1992“ ist eher gering und zwischen „1004“ und „2004“ doch enorm. Außerdem ist die stelleninhärente Approximation ebenfalls eine andere: Wenn man annimmt, dass ein falscher Buchstabe mit einem Fehlermaß des Wertes 1 belegt wird, so ist das Wort womöglich nicht mehr im Wortschatz der Sprache enthalten und ‚falsch'. Eine Jahreszahl wird anders,hingegen nie falsch sein.

Es ist durchaus denkbar, neben dem Wort- und dem numerischen Index einen dritten Typ zu entwickeln: Ein geographischer Index könnte auf Koordinaten Bezug nehmen und eine geographische Näherung bereitstellen. Man könnte also eine Suche auch auf Dokumente mit einem Bezug auf gewisse Orte eingrenzen. Gerade im Bereich der Wirtschafts- und Sozialgeschichte erscheinen solche Anwendungsgebiete sehr viel versprechend, berücksichtigt man doch, dass in der letzten Zeit sehr viele Ansätze über GIS-Anwendungen vorgestellt worden sind. Besonders zur *XVI International Conference of the Association for History and Computing* 2005 in Amsterdam

[8] Erst jüngst ist bekannt geworden, dass der Datenbankhersteller MySQL eine Schnittstelle für die Einbindung von Storage Engines von Drittanbietern definieren will: Datenbank MySQL will mehr Storage-Engines einbinden. Heise-News, 26.04.2006. Adresse: http://www.heise.de/newsticker/meldung/72417 (letzte Einsichtnahme am 17.05.2006).

konnten viele Projektberichte zu GIS-Themen beobachtet werden.[9] Allerdings wird im Rahmen des Projekts eine derartige Erweiterung aus Kapazitätsgründen nicht verfolgt.

Im Bereich der Stringologie wird das hier offenkundige Problem ‚the nearest neighbour problem' genannt. Allerdings ist nicht nur der nächstbeste Treffer gesucht, sondern alle Stellen im Corpus, die in bestimmten Editierdistanz-Grenzen die in der Anfrage genannten Anforderungen erfüllen. Die Approximation darf hier nicht wie in den meisten der bislang in der Literatur beschriebenen Approximationsverfahren auf einer linear vergleichenden Suche aufbauen.[10]

Das Ergebnis ist meist eine längere Ergebnisliste, wobei die Errechnung des Rankings in zwei Stufen erfolgt: Die erste Sortierung der Dokumente erfolgt dokumentenspezifisch, wobei bereits zum Zeitpunkt der Befüllung im Index anhand von Berechnungen zur Relevanz eines Terms in Bezug auf den Container, das Gesamtdokument und den Gesamtcorpus mittels TF/IDF-Verfahren festgelegt wird, wie relevant es in Bezug auf andere Container ist.

Es entsteht also bereits zum Zeitpunkt der Befüllung (‚Feeding') ein grobes Ranking innerhalb kriteriengleicher Ablagen auf der Platte. In einem zweiten Schritt, der erst zum Zeitpunkt der Anfragebearbeitung erfolgt, wird anhand eines Gewichtungsprofils von Containertypen errechnet, wie die Treffermenge sortiert werden muss. Das Gewichtungsprofil entspricht den Anforderungen der unterschiedlichen Interessenlagen der Benutzer. Hier werden die Interessen des Historikers, des Linguisten oder des Diplomatikers abgebildet. Die bezeichneten Profile sind jederzeit änderbar, ohne dass der Index neu erstellt werden muss.

Da vorab nicht bekannt sein kann, für welche Dokumente die Sucharchitektur als solche eingesetzt werden wird, haben wir uns entschieden, dem Beispiel der FAST-Suchmaschine folgend einen modularen Verarbeitungsansatz zu wählen. Es ist dem Betreiber überlassen, die Dokumentenverarbeitung so zu konfigurieren, wie es den Eigenschaften des Corpus entspricht. Besonders die Informationsextraktion, also die automatische Iden-

[9] Vgl. Royal Netherlands Academy for Arts and Sciences. Humanities, Computers and Cultural Heritage. Proceedings of the XVI International Conference of the Association of History and Computing. 14–17 September 2005, Amsterdam. Adresse: http://www.knaw.nl/publicaties/pdf/20051064.pdf (letzte Einsichtnahme am 17.05.2006).

[10] Zu linearen String verarbeitenden Algorithmen: Maxime Crochemore und Wojciech Ritter: Jewels of Stringology. New Jersey 2003. Darin besonders S. 183–198.

tifikation von Entitäten, kann mit Modulen erfolgen, die je nach Bedarf konfiguriert oder gar vom Betreiber selbst programmiert sein können. Die Dokumentenverarbeitung umfasst dabei üblicherweise Schritte wie die Feststellung des Encodings des zu verarbeitenden Dokuments, eine Konvertierung in Unicode/UTF-8, die Tokenisierung und, sofern gewünscht, eine Identifikation von weiteren Strukturmerkmalen wie Satzgrenzen oder Entitäten, wie sie durch lokale Grammatiken erkennbar sind.

Im Rahmen der Pipeline wird auch festgestellt, ob eine Entität nicht im alphabetischen Standardindex, sondern im numerischen oder gegebenenfalls im zukünftig optionalen Geo-Index abgelegt werden soll. Die Typisierung der Entität erfolgt in jedem Fall während der Dokumentenverarbeitung und damit außerhalb der eigentlichen Indexstruktur. Damit bleibt der Indexer selbst schlank und performant.

Der Indexer unterscheidet zwischen den jeweiligen getypten Content- und den Strukturanteilen. Für Strukturanteile wird eine fortlaufende Nummer vergeben, wobei diese bei einer fehlenden Strukturinformation, also bei flachen unstrukturierten Dokumenten, stets null ist. Das bedeutet, dass die Indexerarchitektur flache und damit nicht ausgezeichnete Dokumente ebenfalls verarbeiten kann. Sofern die Dokumentenverarbeitung ohne eine vorliegende Strukturinformation trotzdem andere Indextypen, also Zahlen oder geographische Informationen, erkennt, so wird die Indexerarchitektur sie problemlos indexieren.

In Bezug auf die historisch-fachinformatische Aufgabenstellung kann darauf hingewiesen werden, dass jegliche anwendungsspezifischen Parameter aus der Konfiguration der Dokumentenverarbeitung sowie der Abfrageverarbeitung resultieren. Die Architektur des Indexers selbst ist generisch und flexibel genug, um vielen Bedürfnissen des modernen Information Retrieval zu genügen.

Es wurde beim Design der Indexarchitektur versucht, ein Optimum an Zugriffsgeschwindigkeit und Skalierbarkeit zu finden. Dies ist durch Verwendung der Methoden von Felix Weigel gelungen, der den ‚ContentAware DataGuide' entwickelt hat. Anders als in seiner Referenzimplementierung haben wir auf eine Datenbank verzichtet und verwenden für die Speicherung der Referenzen auf den Corpus reguläre Dateien und zur Verwaltung dieser die Mechanismen des Dateisystems. Auf diese Weise sparen wir Code zur Verwaltung der Referenzen und die Architektur gewinnt deutlich an Stabilität. Das Maximalmaß der Skalierbarkeit einer Implementierung

liegt bei den Grenzen der Partition, in welcher der Index gespeichert wird. Dabei kann es sich auch um ein logisches Volume handeln, das über mehrere physikalische Volumes verteilt ist. Mit derartigen Verfahren liegt die theoretische maximale Skalierbarkeit des Indexers im unterenTerabyte-Bereich. Tests stehen im Moment noch aus, sind jedoch bereits geplant. Als Testdaten sollen Genom-Sequenzierungsdaten dienen, die in den entsprechenden Größenordnungen als XML-Dateien vorliegen.

Alle Kernkomponenten des Indexers sind streng in C++ und die Einbindung von Modulen ist in Python gehalten. Die in Entwicklung befindliche Suchmaschine instantiiert daher nur verschiedenste Klassen einer Bibliothek und führt sie in einer Applikation zusammen. Auf diesem Weg ist es möglich, die Klassen als Bibliothek gesondert abzulegen: unter Unix als ‚Shared Object' und unter Windows als DLL. Die Entwicklungsplattform ist jedoch Linux/Unix.

Von den großen, kommerziellen Vorbildern unterscheidet sich unser Projekt vor allem dadurch, dass Elemente fehlen, die für eine kommerzielle Infrastruktur typisch sind: Wir beabsichtigen, uns auf die Fragestellungen aus der Indexierung zu konzentrieren und überlassen die Implementierung von proaktiven Monitoringsystemen sowie von Ansätzen der verteilten bzw. clusterorientierten Architektur den Benutzern der Bibliothek, die derartige Eigenschaften benötigen.

4. Architekturdetails

Das Projekt hat sich zum Ziel gesetzt, eine Indexierungsarchitektur zu entwickeln, die Content-and-Structure(CAS)-Anfragen verarbeiten kann, wobei der Content-Anteil approximativ und getypt verarbeitet werden kann. Die Architektur soll modular, flexibel, skalierbar, robust und portierbar sein. Ein großer Teil der verwendeten und eingebundenen Technologien entstammen unserem Institut, die Architektur der Dokumentenverarbeitung ist an jene der Altavista- und der ‚FAST-DataSearch'-Engine angelehnt.

Im klassischen XML-Retrieval wird jeweils der Strukturanteil einer Anfrage einerseits und der Contentanteil der Anfrage andererseits ausgewertet. Zur Laufzeit der Anfragebearbeitung wird ein Join der beiden Ergebnismengen berechnet, wobei die beiden Ergebnismengen erst vom Plattenspeicher in den Arbeitsspeicher geladen werden müssen. Felix Weigel, auf dessen An-

sätzen wir unsere Architektur maßgeblich begründen, kritisiert zurecht, dass beim Laden der Ergebnismengen unnötig viele ‚false positives' geladen werden müssen. Außerdem ist richtig, dass die Berechnung des Joins zur Laufzeit der Anfragebearbeitung die Ausgabe des Ergebnisses massiv und unnötig verzögert. Sein Konzept eines ‚Content-Aware DataGuide' nimmt die Berechnung des Joins vorweg und bietet über einen Government- bzw. Containment-Test die Möglichkeit, die Erfüllbarkeit der Anfrage zu prüfen, noch bevor von der Platte Locations geladen werden.[11]

Allerdings überlässt die Referenzimplementierung Felix Weigels die Speicherung der Daten über eine JDBC-Schnittstelle einer Datenbank. Wie oben bereits kurz ausgeführt, sehen wir in diesem Ansatz drei deutliche Nachteile: Falls eine JDBC-Implementierung verwendet wird, die auf einem Netzwerk-Socket basiert, so wird die Zugriffsgeschwindigkeit durch die Abarbeitung der Netzwerkroutinen unnötig verzögert. Dies ist auch der Fall, wenn über das Loopback-Interface auf den Localhost zugegriffen wird. Hier entfällt natürlich die sonst zudem einzuberechnende Latenz bei Netzwerkverbindungen.[12] Der zweite Nachteil liegt darin, dass die Matchingfunktionen der Datenbank überlassen bleiben. Eine Approximation über die Möglichkeiten des LIKE-Operators hinaus ist außerhalb des SQL92-Standards kaum denkbar, will man sich nicht an einen bestimmten Datenbankhersteller binden. Es ist zwar richtig, dass verschiedenste Datenbankhersteller Operatoren entwickelt haben, deren Flexibilität weit jenseits derer des LIKE-Operators liegen. Aber die Modifikation der Matching-Engine einer Datenbank insgesamt erschien uns kaum kalkulierbar. Als dritten Nachteil sehen wir, dass die Verwendung der SQL-Schnittstelle insgesamt unnötig ist. Datenbanken sind generische Werkzeuge zur Speicherung von Daten. Auch wenn die Datenbanktechnologie sehr weit fortgeschritten ist und enorm leistungsfähige Verfahren zur Verwaltung von Daten hervorgebracht hat, so bindet einerseits die Formulierung, andererseits das Parsing und die Verarbeitung von SQL-Ausdrücken Prozessorzeit. Wir sind der Überzeugung, dass betriebssystemnähere und problemorien-

[11] Felix Weigel, Holger Meuss, François Bry und Klaus Schulz: Content-Aware DataGuides: Interleaving IR and DB Indexing Techniques for Efficient Retrieval of Textual XML Data. In: Advances in information retrieval. Proceedings of the 26th European Conference on Information Retrieval (ECIR). Ed. by Sharon McDonald und andere. Berlin, Heidelberg 2004. S. 378–393.

[12] Adresse: http://java.sun.com/products/jdbc/ (letzte Einsichtnahme am 17.05.2006). Offizielle JDBC-Homepage der Firma SUN Microsystems.

tierte Ablagestrukturen gegenüber dem Einsatz von Datenbanktechnologien deutliche Geschwindigkeitsvorteile bieten.

In der Implementierung der Ansätze aus dem CADG unter Verzicht auf eine relationale Datenbank und mit einer eigenen Verwaltung der Content-Tokens sehen wir die Möglichkeit, eine weitere Technologie aus dem Umfeld unseres Instituts zum Einsatz zu bringen: Stoyan Mihov und Klaus Schulz haben ein Verfahren vorgestellt, mit dem in einem Wörterbuch-Automaten sehr schnell approximativ gesucht werden kann.[13]

Die Flexibilität der Architektur wird dadurch sichergestellt, dass der größte Teil der Vorverarbeitung der Dokumente durch Module erfolgt, die der Betreiber der Infrastruktur nach seinen Bedürfnissen konfigurieren kann. Die Schnittstelle zu den Modulen ist genauso wie bei ‚FAST Data-Search' in der Scriptsprache Python gehalten.

Die Relevanzsortierung ist ein Thema, das sich wie natürlich an die Fragestellung der Indexarchitektur anschließt. So verwundert es nicht, dass sich der Autor des ‚Content-Aware DataGuide', Felix Weigel, ebenfalls mit dieser Frage befasst hat.[14] Der Inhalt seiner Überlegungen betrifft die Frage, wie verschiedene Rankingmodelle mit dem Verfahren des vorberechneten Joins zwischen Struktur- und Contentinformation integriert werden können. Seine Betrachtungen beziehen sich dabei auf XPRES[15], BUS[16], XIRQL[17] und XXL[18].

---

[13] Stoyan Mihov und Klaus Schulz: Fast Approximate Search in Large Dictionaries. In: Computational Linguistics 30 (2004). S. 451–477.

[14] Felix Weigel, Holger Meuss, Klaus Schulz und Francois Bry: Content and Structure in Indexing and Ranking XML. In: Proceedings of the 7th International Workshop on the Web and Databases (WebDB 2004). 17–18 June 2004, Paris, France. Ed. by Sihem Amer-Yahia und Luis Gravano. O.O. O. J. S. 67–72..

[15] Jens E. Wolff, Holger Flörke und Armin B. Cremers: Searching and Browsing Collections of Structural Information. In: Proceedings of the IEEE Forum on Research and Technology Advances in Digital Libraries. Washington 2000. S. 141–150.

[16] Dongwook Shin, Hyuncheol Jang und Honglan Jin: BUS. An Effective Indexing and Retrieval Scheme in Structured Documents. In: Proceedings of the 3rd International Conference on Digital Libraries. Pittsburgh 1998. S. 235–243.

[17] Norbert Fuhr und Kai Großjohann: XIRQL: A Query Language for IR in XML Documents. In: Proceedings of the 24th Annual International Conference on Research and development in Information Retrieval. ACM. New York 2001. S. 172–180.

[18] Anja Theobald und Gerhard Weikum: The Index-based XXL Search Engine for Querying XML Data with Relevance Ranking. In: Proceedings of the 8th International Conference on Extending DB Technology. Prag 2002. S. 477–495.

Die atomare Einheit in unserem Index und damit das Objekt, das der Benutzer zu finden gedenkt, ist ein spezifischer Token in einem spezifischen Container. Eigentlich soll in der Regel eine Location gefunden werden, also eine Stelle in einem Text. In Bezug auf semistrukturierte Dokumente kann jedoch nicht mehr von der semantischen Homogenität eines Dokuments ausgegangen werden, und so nimmt der XML-Container mit einem bestimmten Pfad die Stelle ein, die in der Bewertung flacher Corpora das Dokument hat. Es soll also nach einer Stelle in einem bestimmten Container gesucht werden. Da die Stelle jedoch aus einer Kombination von Dokument und XML-Pfad besteht, muss jede Stelle eine Nummer enthalten, damit nicht nur zwischen den Dokumenten im Corpus, sondern zwischen allen Containern unterschieden werden kann.

Die Anfrage sucht stets nach den Containern, in denen ein bestimmter Content eine herausragende, bezeichnende Bedeutung hat. Insofern muss beim Befüllen des Index und beim Erstellen der Rankingparameter bedacht werden, ob ein bestimmter Token in einem bestimmten Container relevant ist. Dies kann durch die Berechnung der ‚Term Frequency' erfolgen. Auf diese Weise ist es möglich, je Dokument die Reihenfolge der relevantesten Wörter festzulegen.

Allerdings kann erst dann eine wirksame Aussage über die Relevanz eines Wortes in Bezug auf den Text getroffen werden, wenn die Termfrequenz zur Häufigkeit des Wortes im Corpus in Bezug gesetzt wird. Diese Bezugsetzung erfolgt rechnerisch über eine Invertierung, der ‚Inverse Document Frequency'.

In Bezug auf semistrukturierte Dokumente muss daher die Relevanz immer in Bezug auf denselben Containertyp ermittelt werden, da es sich bei einem semistrukturierten Corpus genau genommen nicht um einen homogenen Pool an Texten handelt, sondern um eine Menge sehr spezifisch ausgezeichneter Texte, die in den jeweiligen Containern enthalten sind. Aus diesem Grund modifizieren wir die Formel ‚TF/IDF' zu ‚TF/ICF' (Inverse Container Frequency).

Der Wert, der sich aus der Berechnung von TF und ICF ergibt, ist das Sortierungskriterium der Location-Liste in den LL-Dateien. Er wird zusammen mit den Location-IDs abgespeichert, um eine Einfügung an einer entsprechenden Stelle zu erlauben, wenn weitere Daten im Index abgelegt werden.

Für die Berechnung von TF/ICF es ist nötig, die Containerfrequenz des jeweiligen Wortes zu kennen. Daher steht sie an der ersten Stelle der Location-Liste. Beim Einfügen eines neuen Eintrags muss die Location-Liste ohnehin geladen werden, und so kann ohne großen gesonderten Aufwand die Information über die ICF des jeweiligen Wortes gleich in den Speicher geladen und zur Verarbeitung bereitgestellt werden.

In Bezug auf die von Felix Weigel betrachteten Rankingverfahren lehnt sich unser Verfahren an BUS an. Allerdings trennen wir klar zwischen der Relevanzfeststellung des Contentanteils und des Strukturanteils, da letzteres in unserer Gesamtarchitektur eine besondere Rolle spielt: Die Bewertung des Strukturanteils fließt zum Zeitpunkt der Befüllung noch nicht mit ein, sondern über ihn bilden wir die Präferenzen der verschiedenen Benutzergruppen ab. Doch dazu später mehr im Abschnitt über das Ranking.

Die Komponenten der Architektur sind in einzelnen:

- der Feeder, welcher folgende Komponenten enthält:
  - den Parser,
  - die Document Processing Pipeline,
- der Indexer (apCADG), welcher folgende Komponenten enthält:
  - den DataGuide,
  - den WordIndex,
  - den NumericIndex,
  - als Option den Geo-Index,
  - das Lametta,
  - die StorageArea auf der Platte,
  - den QueryProcessor,
- der QueryHandler, der folgende Komponenten enthält:
  - die Query Processing Pipeline,
  - den Kontrollautomaten-Compiler,
  - den Ranker.

Die wichtigsten der genannten Komponenten sollten folgend kurz vorgestellt werden:

4.1 Der Feeder

Der Feeder ist eine Wrapper-Klasse, die das einzulesende Dokument entgegennimmt und dem SAX-Parser übergibt. Dieser zerlegt es in Pfad-Con-

tent-Tupel und übergibt jeden der Tupel an die Document Processing Pipeline. Diese implementiert eine in den C++-Code eingebettete Python-Instanz, wobei die Tupel an die Python-Instanz zur Verarbeitung übergeben werden. Die Instanz lädt je nach Konfiguration ein Framework an Python-Modulen, deren wichtigste Aufgaben folgende sind:

- Encoding-Normalisierung auf UTF-8,
- Tokenisierung,
- Annotation von Entitäten.

Es sind zahlreiche weitere Bearbeitungsstufen denkbar wie eine Language-Detection, ein Lemmatizer, ein Vectorizer, eine Date/Time-Extraction und viele mehr. Wir beschränken uns auf letztere, da sie wirklich essentiell sind. Die Abwendung von der Entität der XML-Datei als solcher ist dadurch gerechtfertigt, dass einerseits eine XML-Datei mehrere Quellentexte enthalten kann, und der Benutzer andererseits nicht daran interessiert ist, Dateien, sondern Dokumente, respektive Textstellen, im Corpus zu finden.

4.2 Der DataGuide

Man kann ein XML-Dokument als Baum darstellen, bei dem der äußerste XML-Knoten der Wurzelknoten und jeder innere Container ein entsprechender Kind-Knoten ist.[19] Wenn man die Wurzelknoten aller Dokumente im Corpus zusammenfasst, und dabei alle namensgleichen Container konsolidiert, so erhält man einen Baum an all den XML-Pfaden, die im Corpus enthalten sind. Ein Baum dieses Typs wird DataGuide genannt. Jeder Knoten dieses Baumes erhält dabei eine ganzzahlige, fortlaufende Nummer, ungeachtet dessen, ob es sich dabei um einen Knoten des Schema-Typs PCDATA (Knoten ohne weitere Kinder) handelt oder nicht.

Unsere Implementierung verwendet eine modifizierte Fassung der ‚core::tree'-Template-Klasse von Justin Gottschlich[20], die insofern verändert wurde, als dass die Suche eines bestimmten Kindknotens im Baum nicht mehr linear, sondern mit einer STL-Map logarithmisch erfolgt. Wir haben

[19] Zur Baumdarstellung von XML-Dokumenten siehe: Jamens Clark und Steven deRose: XML Path Language (XPath) 1.0. W3C Recommendation 16 November 1999. Adresse: http://www.w3c.org/TR/xpath (letzte Einsichtnahme am 17.05.2006).

[20] Vgl. Justin Gottschlich: C++ Trees. Adresse: http://www.gamedev.net/reference/programing/features/coretree1/ (letzte Einsichtnahme am 17.05.2006).

das C++-Template so verändert, dass jeder Elternknoten seine Kindknoten anhand eines eineindeutigen Namens identifiziert. Der Name ist dabei der Tag des entsprechenden XML-Containers.

Besondere Beachtung verdienen Attribute, denn sie kommen unter Umständen mehrfach in demselben XML-Tag vor. Wir behandeln Attribute wie Pfaderweiterungen, wobei der Inhalt des Containers bei mehreren Attributen mehrfach indexiert wird. Zudem wird die Gesamtheit aller Attribute zusammengefasst und Container auf diese Weise anhand ihrer gesamten Strukturidentifikation auffindbar gemacht.

4.3 Der WordIndex und Kompilierungsverfahren

Der Zugriff auf den Wortindex implementiert ein Verfahren nach Klaus Schulz und Stoyan Mihov,[21] nach dem mittels Backtracking in einem Wortindex einerseits und einem Levenshtein-Kontrollautomaten andererseits bei kurzen Wörtern und einer niedrigen Editierdistanz drastische Geschwindigkeitsgewinne erreicht werden können. Das Verfahren setzt also einen bereits fertig kompilierten Wortindex voraus, dessen Kompilierung folgend kurz beschrieben werden soll.

In seiner logischen Darstellung besteht der Index aus einem klassischen Trie.[22] Von einer Wurzel ausgehend führen deterministische Übergänge zu neuen Zuständen, wobei bei jeder tieferen Übergangsstufe der Präfix des eingegebenen Wortes weiter spezifiziert wird. Mathematisch wird eine derartige Struktur als ein deterministischer, endlicher azyklischer (gerichteter) Graph bezeichnet.

Würde man nun jeden Zustand des Graphen im Speicher eigens allokieren, so bräuchte man ohne Rücksicht auf die im jeweiligen Zustand real existierende Anzahl an Übergängen eine linear große Speichermenge. Da jedoch die Anzahl der Wörter eine der stark skalierenden Größen im Index ist, wird man bestrebt sein, gerade hier den Speicherverbrauch einzudämmen. Aus diesem Grund liegt es nahe, eine Kompression der Übergangstabelle nach einem Verfahren von Tarjan vorzunehmen.[23] Dieses Verfahren

[21] Vgl. Anm. 13.

[22] Zu Tries siehe: Donald E. Knuth: The Art of Computer Programming. Vol. 3, 2nd ed. Boston 1998. S. 492–512.

[23] Robert Endre Tarjan und Andrew Chi-Chih Yao: Storing a sparse table. In: Communications of the ACM 22 (1979). S. 606–611.

verschränkt Übergänge in einer Weise, in der die Performanz im Zugriff auf die Zustände nicht beeinträchtigt wird, wobei aber durch die Verschränkung enorm an Platz gespart wird.

In unserem Ansatz verwenden wir das Tarjan-Verfahren, jedoch erweitern wir es zu einem Transduktor, der nicht nur Eingabewörter erkennt, sondern bei einem Endzustand einen Ausgabewert zurückgibt. Der Ausga-bewert in unserem WordIndex besteht aus einer Sprungadresse im Speicher und einer fortlaufenden Zahl. Die Sprungadresse zeigt auf den entsprechenden Bitstring in der Lametta-Datenstruktur. Die fortlaufende Zahl wird für die Adressierung der jeweiligen Location-Datei auf der Platte benötigt.

Der Kompilierungslauf ist in diesem Sinne eigentlich ein Kompressionslauf, der den Stack und die Liste der bereits verarbeiten Wörter zusammenführt und sortiert und anschließend daraus den Trie in seiner kompakten Form berechnet und im Speicher zur Verfügung stellt. Dieselben Verfahren kommen bei der Kompilierung der Lamettatabelle zum Einsatz.

Der Tarjan-Mechanismus allokiert also je Zeile der Übergangstabelle entsprechend der Länge des Alphabets in einem vordefinierten und mit Null-Pointern initialisierten Speicherbereich 2n+1 Zellen, wobei n die Mächtigkeit des Alphabets ist. Eine Zelle beinhaltet zwei Zeiger, wobei der eine auf den nächsten Zustand und der andere auf den Anfang des aktuellen Zustands zeigt, um sicherzustellen, dass der aktuelle Übergang auch ein Übergang des aktuellen Zustands ist. In Erweiterung zum Tarjan-Schema fügen wir dem Algorithmus noch einen weiteren Zeiger hinzu: Am Anfang der Tarjan-Zeile befindet sich in unserer Implementierung ein Zeiger auf den Anfang des vorhergehenden Zustands, welcher benötigt wird, um mit einer möglichst geringen Speichersignatur eine Rückwärts-Auflösung des Tokens zu ermöglichen.

4.4 Der NumericIndex

Der Index zur numerischen Approximation bewegt sich lediglich in einer einzigen, zeitlichen Dimension. Eine Approximation erfolgt daher ausschließlich numerisch. Die am besten geeignete Datenstruktur hierfür ist eine Double Ended Queue: Eine zweifach verlinkte Kette der C++ Standard Template Library, die einen Zugriff über einen Index erlaubt, aber auch Einfüge-Operationen zur Verfügung stellt. Jedes Glied der Kette ist dabei vergleichbar mit einem Endzustand und beinhaltet daher einen Zeiger auf

die entsprechende Lametta-Zeile sowie eine Zahl als Indexmerkmal für die Datenstruktur auf der Platte. Diese Zahlen sind über alle Index-Systeme hinweg fortlaufend.

Zum gegenwärtigen Stand sehen wir den Einsatz eines Konstrukts aus der STL als den effektivsten Weg, zu einer funktionierenden Architektur zu gelangen, und wollen daher nicht ausschließen, dass hier noch Optimierungspotential besteht.

4.5 Das Lametta

Das Lametta hat die folgende Funktion: Die Nummer des XML-Knotens, die der DataGuide bei der Queryverarbeitung liefert, dient als Index für eine Bitfolge, bei der jeweils das erste Bit den ‚Containment-Test', und das jeweils zweite Bit den ‚Government-Test' ermöglicht. Der Containment-Test beantwortet die Frage, ob ein XML-Knoten selbst das gesuchte Wort enthält. Der Government-Test beantwortet die Frage, ob eines der Kinder des aktuell betrachteten Knotens das Wort enthält. Der Containment-Test gibt die qualifizierte Antwort, ob es gerechtfertigt ist, mit der Nummer des DataGuide-Knotens und der Nummer des Worts aus dem Wortindex auf der Platte aus der ‚Ablagestruktur nach Contentinformation' die Liste der zutreffenden Locations zu holen.

Der Government-Test ist besonders bei Anfragen wichtig, bei denen sich in der Pfadinformation ein Freiheitsgrad im Sinne eines Kleene-Sternchens befindet. Hier muss der Pfadbaum in der Tiefe durchlaufen werden, um nicht Pfade zu selektieren, bei denen das gesuchte Wort ohnehin nicht enthalten ist. Wir verwenden den binären Government-Test um zu klären, ob sich ein Hinabsteigen in einen bestimmten Pfad überhaupt lohnt. Die konzeptionelle Idee hierfür entstammt den Arbeiten von Felix Weigel.

Unsere Implementierung ohne Rückgriff auf eine Datenbank arbeitet mit festen Bitstring-Längen, wobei im Gegensatz zur Kompilierung von Automaten keine Übergänge, sondern lediglich eine Bool'sche Wahr- oder Falsch-Information kodiert werden soll. Beide Informationen werden durch Gerichtetheit des Speicherwertes auf den Anfang oder das Ende der festen Zeilenlänge der Tabelle kodiert. Auf diese Weise lässt sich eine ebenso leistungsfähige, Speicher sparende Kompression wie bei der Kompilierung des WordIndex erreichen.

Sollte bei einer Neukompilierung die feste Länge des Bitstrings nicht ausreichen, so wird diese um einen festen Wert erweitert. Dieses Verfahren belegt angesichts der Tarjan-Kompressionsmethode keinen unnötigen Speicherplatz, da unbenutzte Speicherzellen weiterhin von anderen passenden Bitstrings benutzt werden können.

Die Abarbeitung der beiden Tests erfolgt, indem aus dem WordIndex auf den Speicherplatz des entsprechenden Lametta-Strings gesprungen wird. Mit der Indexzahl aus dem DataGuide wird die Position der jeweiligen Speicherzelle ermittelt. Wenn der Zeiger in dieser Zelle mit dem Zeiger auf die Lamettafolge, also die erste Speicherzelle der Zeile, identisch ist, so ist der Test positiv. Zeigt der Zeiger in der Speicherzelle auf das Ende der Lamettafolge, so ist der Test negativ. Nicht beschriebene Zellen enthalten den Null-Wert.

4.6 Die StorageArea

Zur Ablage der Locations auf der Festplatte haben wir folgende Systematik erarbeitet: Um den Durchgriff auf die Daten bei der Suche möglichst robust, schnell, skalier- und portierbar zu halten, haben wir uns an den Möglichkeiten orientiert, die gängige Betriebssysteme zur Verfügung stellen. Die Ansprache einer Locations-Datei erfolgt daher über einen Dateipfad, der nach der Abarbeitung der im Speicher befindlichen Datenstrukturen zusammengestellt wurde. Die fortlaufenden Nummern aus dem DataGuide und den Indexstrukturen werden in ihre hexadezimale Repräsentation konvertiert und in jeweils zwei Zeichen gestückelt. Anschließend wird eine Datei mit dem Namen „LL“ hinzugefügt, sodass ein Pfad wie folgt aussieht:

Aus einer DataGuide-Nummer „3f4d“ und einer Wordindex-Nummer „19bc3d“ wird nun der Pfad „<Stamm>/3f/4d/19/bc/3d/LL“. „LL“ ist dabei eine Datei, die die Liste aller Locations enthält. Der Dateiname wurde aus dem Grund auf „LL“, bzw. auf „ZZ“ gesetzt, weil diese Buchstaben nicht Teil des hexadezimalen Alphabets sind.

Bei einer hexadezimal dreistelligen Wortadressierung liegt die maximale Skalierung bei 16 777 216 möglichen Wörtern. Falls diese Anzahl übertroffen wird, so wird der bestehende Baum in ein neu zu schaffendes Unterverzeichnis „<Stamm>/00“ kopiert und im neu anzulegenden Verzeichnis „<Stamm>/01“ fortgesetzt. Um eine entsprechende Skalierung zu gewähr-

leisten, ist es also angebracht, von vornherein einen ausreichend großen Adressraum zur Verfügung zu stellen.

Die Aufteilung des Pfades in jeweils zwei Stellen erfolgt aus dem Grund, dass viele Dateisysteme nicht in der Lage sind, größere Mengen an Dateien in einem Verzeichnis in logarithmischer Zeit zu verwalten. Ein einfaches „ls“ in einem ext2-Dateisystem und 100 000 Dateien in einem Verzeichnis führt zu einer beachtlichen Wartezeit. Mit Blick auf die geforderte Portierbarkeit muss von anderen, noch deutlich weniger leistungsfähigen Dateisystemen wie FAT32 ausgegangen werden. So befinden sich in jedem Verzeichnis lediglich maximal 256 Unterverzeichnisse – eine Zahl, die jedes Dateisystem mit ausreichender Performanz verwalten kann.

Die Ordnung nach Pfad-Wort ist aus dem Grund so gewählt, weil es XML-Tags gibt, die keinen Content enthalten. Sie haben keinen schließenden Tag und stehen für sich. In diesem Fall ist der Pfad zur „LL“-Datei: “<Stamm>/3f/4d/00/00/00/LL“. Mit einer Anordnung, in welcher der Pfad zu einer Pfadnummer, der mangelnde Content aber nur zu einer „0“ führt, sind derartige Pfadelemente ohne Probleme abbildbar.

Die eingangs geforderte Eigenschaft, dynamische ‚Drilldowns‘ anzubieten, wird mit folgendem Mechanismus unterstützt: Ein Drilldown besagt, dass in der Ergebnismenge eines Suchworts alle erkannten Entitäten dynamisch und in Abhängigkeit vom Suchwort ausgegeben werden sollen. Wenn beispielsweise nach „HSV“[24] gesucht wird, sollen alle Namen der Fußballspieler ausgegeben werden, die ebenfalls, und zwar ausschließlich, in den Dokumenten vorkommen, in welchen der gesuchte String (hier „HSV“) vorkommt. In Bezug auf semistrukturierte Dokumente bedeutet das, dass der Spielername in der Dokumentenverarbeitung gesondert als solcher ausgezeichnet wird und die Wortindex-Nummer aller Spielernamen in demselben Verzeichnis, in welchem sich die „LL“-Datei befindet, in einer „ZZ“-Datei verzeichnet wird. Entsprechend der Häufigkeit kann hier bereits eine erste Sortierung vorgenommen werden.

Übertragen auf die historische Fragestellung und unter der Voraussetzung, dass eine Eigennamenerkennung in der Dokumentenverarbeitung zuverlässige Ergebnisse liefert, kann damit etwa die Frage beantwortet werden, welche Personen in einem bestimmten historischen Kontext auftauchen. Falls man eine Ortsnamenerkennung implementiert, kann in Bezug auf einen bestimmten historischen Vorgang ein Profil aller Orte erstellt

[24] Hamburger Sportverein.

werden, zu denen der Vorgang in Bezug steht. Ein Drilldown würde nun in der ersten Ergebnisdarstellung die erkannten Entitäten in Bezug auf ein Suchwort ausgeben.

Der Benutzer würde in einem nächsten Schritt den ausgewählten Begriff zusammen mit dem ursprünglichen Suchwort als erneute Suche abschicken. Ab hier handelt es sich um Suchvorgänge mit mehreren Constraints, sodass Joins unvermeidlich sind. Die Bearbeitung von Joins von Ergebnismengen ist jedoch ein weiterführendes Thema und soll an dieser Stelle explizit als offener Punkt vermerkt werden. In dieselbe Problematik fällt auch die der Verarbeitung von Seitenbedingungen, die in XPath spezifiziert sind.

Wichtig in Bezug auf die Erfassung von Drilldowns im Index ist, dass die Auszeichnung einer Entität außerhalb des Index erfolgt. Der Index selbst kann ohne eine erforderliche Neukonfiguration und ein erneutes Einlesen des gesamten Corpus ohne Unterbrechung weiter betrieben werden. Die Auszeichnung erfolgt in einem Modul in der Dokumentenverarbeitung und kann beliebig vom Benutzer konfiguriert werden.

4.7 Der QueryProcessor

Der QueryProcessor übernimmt vom QueryHandler, der Steuerkomponente der Query-Verarbeitung, eine Strukturinformation und ein Eingabewort in Form eines standardkonformen XPath-Ausdrucks[25] sowie einen auf dem Eingabewort fertig kompilierten Kontrollautomaten, der implizit die maximal erlaubte Approximationsdistanz bereits enthält. Seine Aufgabe ist, die Anfrage auf der Basis der zuvor vorgestellten Datenstrukturen abzuarbeiten und eine Liste an treffenden Locations zurückzugeben, wobei jede Location mit dem Multiplikator der approximativen Distanz, aber auch den TF/ICF-Werten annotiert wird. Die Auswertung der für das Ranking relevanten Informationen erfolgt durch den Ranker.

Nach der Übernahme des XPath-Ausdrucks wird dieser in seinen Content- und seinen Strukturanteil aufgetrennt. Zudem wird ermittelt, ob der Strukturanteil Freiheitsgrade enthält oder nicht.

Der mit übergebene Automat dient als Kontrollsystem bei der Bearbeitung des WordIndex, wobei in beiden Strukturen parallel in der Tiefe fortgeschritten wird. Falls ein Zeichen aus dem Eingabewort nicht weiter im

[25] Der XPath-Standard. Adresse: http://www.w3c.org/TR/xpath (letzte Einsichtnahme am 17.05.2006).

WordIndex vorhanden ist, wird anhand des Kontrollautomaten festgestellt, ob die bisher vorgenommenen Abweichungen noch innerhalb der maximal zulässigen Editierdistanz liegen. Wenn dies der Fall ist, kann im WordIndex nach einem alternativen Übergang gesucht und dieser Weg weiter verfolgt werden. Ein Treffer ist dann gefunden, wenn sowohl im Kontrollautomaten als auch im WordIndex ein Endzustand erreicht wird. Allerdings endet die Suche in diesem Fall nicht nach dem Finden eines ersten Treffers, sondern mittels Backtracking wird versucht, den gesamten Suchraum innerhalb der definierten Fehlertoleranz auszuschöpfen.

Da ein Treffer im WordIndex zu einem Lametta-Bitstring führt, ist nun auch eine Abarbeitung der Überprüfung des Strukturanteils möglich. Falls der Strukturanteil keine Freiheitsgrade aufweist, so kann direkt mit einem Containment-Test ermittelt werden, ob die Anfrage erfüllbar ist. Falls Freiheitsgrade vorhanden sind, so wird anhand des Government-Tests im Lametta-Bitstring untersucht, wie weit in den Dataguide-Baum hinabgestiegen werden kann. Ein positiver Government-Test besagt, dass Kinder des aktuell betrachteten Knotens im Corpus das besagte Suchwort enthalten. Bei jedem Kind wird dann wiederum ein Containment-Test vorgenommen, um Knoten zu identifizieren, in denen im Corpus das zuvor identifizierte Wort auch wirklich vorkommt.

Führen Government- sowie Containment-Tests anhand des Lametta-Bitstrings schließlich zu Knoten, mit denen die Anfrage erfüllbar ist, so wird aus den von den beiden Datenstrukturen erhaltenen Nummern der Pfad zur Locations-Datei zusammengestellt. Die Locations werden zusammen mit der erreichten Editierdistanz und mit den einzeln verfügbaren TF/ICF-Werten an den Ranker gegeben.

4.8 Die Query Processing Pipeline

Grundsätzlich ist jede Indexierung ein symmetrisches Verfahren: Damit die Dokumente in einem Corpus gefunden werden können, müssen die Texte in die Einheiten zerlegt werden, die später gefunden werden wollen. Also enthält ein Index atomare Informationsentitäten, die aus den zu indexierenden Dateien oder Dokumenten herausgearbeitet werden müssen. Die Symmetrie besteht insofern, als auch die Anfrage an eine Suchmaschine einer Verarbeitung unterworfen sein muss: Die Anfrage muss so weit aufbereitet werden, dass sie den indexierten Entitäten exakt entspricht. Archi-

tektonisch ergeben sich daher zwei Verarbeitungspipelines, eine so genannte ‚Document Processing Pipeline' sowie eine ‚Query Processing Pipeline'.

Der Unterschied zwischen beiden Zugriffsmethoden auf den Index besteht darin, dass die Query Processing Pipeline die Verarbeitung in Echtzeit vornehmen muss, und die gefundenen Ergebnisse noch einer Nachbearbeitung unterworfen werden müssen. In letzterer muss noch eine Gewichtung und die Auswertung der Rankingparameter errechnet werden. Weil die Document Processing Pipeline keinen zeitkritischen Einschränkungen unterworfen ist, da die Dokumentenverarbeitung vorab erfolgt, hängt eine gute Antwortzeit direkt von der Verarbeitungsgeschwindigkeit der Query Processing Pipeline ab. Im Gegensatz zur Dokumentenverarbeitung findet man in Installationen kommerzieller Suchmaschinen bei der Anfrageverarbeitung selten Module in Skriptsprachen. Allerdings sind die Aufgaben auch meist deutlich weniger komplex.

Die Query-Verarbeitung hat die Aufgabe, aus dem eingegebenen String und der in der GUI ausgewählten Strukturinformation alle nötigen Informationen zu extrahieren und diese dergestalt aufzubereiten, dass sie an den Index übergeben werden können. Komponenten einer typischen Query Processing Pipeline sind:

- Encoding-Feststellung,
- Normalisierung auf UTF-8,
- Erkennung der eingegebenen Sprache,
- Tokenisierung,
- Lemmatisierung.

Da die Entwicklung derartiger Module einen nicht zu unterschätzenden Mehraufwand darstellt, soll im Rahmen dieses Projekts lediglich ein Tokenisierungsmodul entwickelt werden. In Bezug auf die übrigen Arbeitsschritte ist die Entwicklung der entsprechenden Module an den Bedarf aus dem zukünftigen Einsatz gebunden. Da während der Entwicklungsphase des vorliegenden Projekts von einer kontrollierbaren Datenlandschaft und sauberen Beispieldaten ausgegangen werden kann und ein entsprechender Bearbeitungsbedarf nicht besteht, werden die Module von passiven Prototypen ersetzt, die später gegen funktionale Komponenten ausgetauscht werden können.

Das Ergebnis der Anfragebearbeitung ist in jedem Fall ein standardkonformer XPath-Ausdruck sowie ein kompilierter Kontrollautomat, der die Information enthält, ob und in welchem Rahmen eine approximative Suche vorgenommen werden muss. Sollte der Eingabestring der Anfrage aus mehreren Wörtern bestehen, so werden im Query-Tokenizer mehrere XPath-Anfragen mit gegebenenfalls mehreren Automaten generiert. Die Schnittstelle zum Index ist explizit konform zum XPath-Standard, um auch maschinelle Anfragen aus anderer Quelle abarbeiten zu können.

4.9 Der Kontrollautomaten-Compiler

Je nach Anforderung und Approximationsbedarf wird nach dem Verfahren von Mihov und Schulz auf jedes Eingabewort ein Automat kompiliert, der gemäß der Definition der Levenshtein-Editieroperationen[26] (Einfügungen, Löschungen, Ersetzungen) die Kontrolle von Abweichungen innerhalb bestimmter Distanzgrenzen erlaubt.[27]
Der Kompilierungsvorgang verläuft jedoch nicht rein algorithmisch, sondern es werden äquivalente Übergänge einkompiliert, sodass etwa Phänomene wie die Äquivalenz von „u“ und „v“ im Anlaut ohne Feststellung eines Fehlers behandelt werden können. Es können so aus der diachronen Linguistik bekannte graphemische (Allographe) oder phonetische Phänomene abgebildet und entsprechend behandelt werden.[28] Ziel hierbei ist die mit Blick auf die Eigenschaften der alten Texte erforderliche Erweiterung des Recall.

4.10 Der Ranker

Wie oben beschrieben, trennen wir zwischen der Bewertung der Contentrelevanz und der Bewertung der Strukturrelevanz: Beinhaltet die Strukturinformation keine Freiheitsgrade, so ist eine Gewichtung der Strukturinfor-

[26] Vladimir I. Levenshtein: Binary codes capable of correcting deletions, insertions, and reversals. In: Doklady Akademii Nauk SSSR 163 (1965). S. 845–848 (Russisch). Englische Übersetzung in: Soviet Physics Doklady 10 (1966). S. 707–710.

[27] Vgl. Kemal Oflazer: Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. In: Computational Linguistics 22/1 (1996). S. 73–89.

[28] Zu den diachronen Varianzen zum und vom Frühneuhochdeutschen in spätere Sprachstände: Gerhard Philipp: Einführung ins Frühneuhochdeutsche. Sprachgeschichte, Grammatik, Texte. Heidelberg 1980.

mation überflüssig, da der Benutzer dem System keinen Entscheidungsspielraum lässt. Beinhaltet die Strukturinformation doch Freiheitsgrade, so erfolgt in unserer Architektur eine benutzerbezogene Relevanzgewichtung: Unter der Annahme, dass es sich um eine zweckgebundene Suchmaschine für eine bestimmte Benutzergruppe und für eine bestimmte Sorte an Dokumenten handelt, sehen wir die Möglichkeit der Gewichtsmultiplikation durch den Ranker vor. Hier genügen wir der Forderung, dass verschiedene Nutzergruppen einer Suchmaschine für historische Daten unterschiedliche Interessenschwerpunkte haben können. Indem abhängig vom Benutzerprofil eine besondere Ergebnisgewichtung vorgenommen werden kann, lässt sich die Ergebnismenge so sortieren, dass beispielsweise eher Treffer aus dem Originaltext oder eher Treffer aus dem Kommentar oder eher Treffer aus den Metadaten angezeigt werden.

In die Errechnung der Sortierung der Locations fließen damit die Editierdistanz aus der Abarbeitung des Automaten, der TF/ICF-Wert sowie die benutzerorientierte Gewichtung der Strukturinformation ein.

5. Projektstand

Zum Zeitpunkt der Kalenderwoche 16/2006 sind der Feeder samt XML-Parser, die Architektur der DocumentProcessing Pipeline, der DataGuide und erste Teile des WordIndex fertig gestellt.