

Text Retrieval

Version: 2007-03-29/1
Arbeitspaket: AP1
verantwortlicher Partner: FH Worms

TextGrid

Modulare Plattform für verteilte und kooperative
wissenschaftliche Textdatenverarbeitung -
ein Community-Grid für die Geisteswissenschaften



Projekt: **TextGrid**

Teil des D-Grid Verbundes und der deutschen e-Science Initiative

BMBF Förderkennzeichen: 07TG01A-H

Laufzeit: Februar 2006 - Januar 2009

Dokumentstatus: final

Verfügbarkeit: öffentlich

Autoren:

Andreas Aschenbrenner, SUB

Christian Graiger, Uni Würzburg

Christoph Ludwig, FH Worms

Wolfgang Pempe, Saphor GmbH

Christian Simon, IDS

Andrea Zielinski, IDS

Inhaltsverzeichnis

1	Einleitung	4
2	Aspekte des Text Retrievals	4
2.1	Text Retrieval aus Anwender-Sicht	4
2.1.1	Suche in Textdaten vs. Metadaten	5
2.1.2	Suche in Freitext vs. ausgezeichneten Texten	5
2.1.3	Nutzerschnittstelle	6
2.2	Text Retrieval aus technischer / technologischer Sicht	8
2.2.1	Werkzeuge zur Speicherung optimal durchsuchbarer Daten	9
2.2.1.1	Relationale Datenbanken	9
2.3	XML-Datenbanken	9
2.3.1.1	Volltextdatenbanken bzw. Volltext-Indizierer	10
2.3.2	Suche über Institutionsgrenzen hinweg	10
2.3.2.1	Zentraler Index	11
2.3.2.2	Metasuche	11
2.3.2.3	Föderierte Suche	12
2.3.2.4	Vorgeschaltete Filter bei verteilten Suchen	12
2.3.2.5	TextGrid Sicht	13
2.3.3	Text Mining Techniken	13
2.3.4	Semantische Suche	14
2.3.4.1	TextGrid Sicht	14
2.3.5	sprachspezifische Suche	15
2.3.6	multilinguale Suche	15
3	Auswahl existierender Werkzeuge	16
3.1	eXist	16
3.2	Berkeley DB-XML	16
3.3	Apache Xindice	17
3.4	Apache Lucene	17
3.5	Suchwerkzeuge in „Der junge Goethe in seiner Zeit“, Internet- und CD-ROM-Version	17
3.6	TEI-Publisher	18
3.6.1	Funktionalität	18
3.6.2	Voraussetzungen	18
3.6.3	Evaluierung	18
4	Zusammenfassung	19
Anhang A:	Literatur	19

1 Einleitung

Einer der großen Vorteile, die TextGrid seinen Nutzern gegenüber herkömmlichen Systemen zur wissenschaftlichen Textverarbeitung bieten soll, ist der Zugang zu enormen Mengen an Textressourcen, die entweder direkt in TextGrid eingestellt wurden oder in angebundnen Archiven bereit stehen.

Allerdings haben bereits die Datenmengen, die ein typischer Anwender auf seinem lokalen Rechner angesammelt hat, ein derart großes Volumen erreicht, dass sich spezifische Dokumente oder gar Textpassagen oft nur noch mit Hilfe von Programmen wie *Google Desktop* oder Apples *Spotlight* wiederfinden lassen. Wenn die Anwender aus den in TextGrid verfügbaren Daten Nutzen ziehen sollen, benötigen Sie also spezialisierte Werkzeuge, um in den TextGrid-Ressourcen zu suchen. Hinzu kommt noch, dass Textwissenschaftler häufig mit Fragestellungen an den Datenbestand herangehen, die nur von einer Volltextsuche, evtl. noch erweitert durch reguläre Ausdrücke und boolesche Verknüpfungen, nicht befriedigend beantwortet werden kann. Orthographische Varianten, Transkriptionen und Transliterationen, Anforderungen an den Kontext der Fundstellen sowie Einschränkungen, die nur bei Auswertung der Metadaten eines Textes berücksichtigt werden können, machen deutlich, dass die Recherchekomponente in TextGrid hochkomplexere Anfragen bedienen muss.

Ziel dieses Berichtes ist, einen Überblick über die Anforderungen an das Text Retrieval in TextGrid sowie über die verfügbaren einschlägigen Programme und Softwarebibliotheken zu geben.

Abschnitt 2 ist zweigeteilt: Abschnitt 2.1 betrachtet die Anforderungen, die Anwender an die Recherche stellen. Dagegen erörtert Abschnitt 2.2 die technologischen Herausforderungen und Optionen. Schließlich werden in Abschnitt 3 existierende Programme und Werkzeuge besprochen, wobei sich dieser Bericht zwangsläufig auf eine Auswahl beschränken muss.

2 Aspekte des Text Retrievals

Die Möglichkeiten, Texte zu recherchieren, werden von den Endanwendern eines Systems meist ganz anders wahrgenommen als von den Entwicklern, Administratoren und Content Providern. Erstere treten mit einer Fragestellung an das System heran, die sie in einer (bezogen auf ihren fachwissenschaftlichen Hintergrund) möglichst natürlichen Weise formulieren möchten. Sie werden die Qualität eines Retrieval-Werkzeugs in erster Linie daran messen, wie einfach sich die Suchanfragen stellen lassen, wie lange die Recherche dauert und wie genau die gelieferte Treffermenge sich mit ihren Erwartungen deckt. Wie die Rechercheergebnisse im Einzelnen ermittelt werden, ist für die Endanwender in aller Regel unerheblich.

Entwickler, Administratoren und Content Provider befassen sich eher mit der Komplexität des Gesamtsystems, seiner Performanz und dem Aufwand für die Aufbereitung und Pflege des Datenbestandes. Hierfür sind nicht zuletzt die genaue Abgrenzung des Funktionsumfangs und die eingesetzten Technologien entscheidend.

Diese fundamental unterschiedlichen Sichtweisen auf das System spiegeln sich auch in den beiden folgenden Abschnitten wider; wir betrachten zunächst verschiedene Typen von Recherchen und die Anforderungen an die Nutzerschnittstelle bevor wir uns den in Frage kommenden Technologien zuwenden.

2.1 Text Retrieval aus Anwender-Sicht

Anwender erwarten, dass ihre Suchanfrage in jedem Fall eine vollständige Liste aller einschlägigen Dokumente im System liefert, möglichst noch mit Stellenangaben innerhalb der Dokumente. Das Rechercheergebnis sollte auch einen Indikator umfassen, wie genau die einzelnen Treffer auf die formulierte Anfrage passen.

Wir diskutieren in den nachstehenden Abschnitten zunächst, welche Datenräume für diese Anfragen grundsätzlich zur Verfügung stehen, bevor wir uns der Gestaltung der Nutzerschnittstelle zuwenden, mit der die Anwender ihre Anfragen möglichst effizient formulieren können.

2.1.1 Suche in Textdaten vs. Metadaten

Grundsätzlich stellt sich bei jeder Recherche die Frage, welche Daten durchsucht werden sollen – interessiert sich der Anwender für Texte, die zwischen 1618 und 1648 entstanden, so wird diese Information im Allgemeinen nicht in den Texten selbst, sondern in den zugehörigen Metadaten vermerkt sein. Interessiert sich ein Anwender hingegen für Texte, die auf den Dreißigjährigen Krieg Bezug nehmen, so wird er die Texte selbst u. a. nach diesen Jahreszahlen durchsuchen müssen.

An diesem Beispiel zeigt sich bereits ein wesentlicher Unterschied zwischen der Suche auf Metadaten und der Suche in den Inhaltsdaten: In letzteren wird die Referenz häufig nur implizit sein; eine Nennung König Gustav Adolfs oder des Westfälischen Friedens im Text kann – muss aber nicht notwendig – darauf hindeuten, dass sich der Text auf den Dreißigjährigen Krieg bezieht. Die Ergebnisse einer Suche in den Inhaltsdaten zu dieser Fragestellung werden also zwangsläufig in ihrer Relevanz stark variieren. Die Qualität der Ergebnisse hängt deshalb sehr stark von der Erfahrung und dem Vorwissen des Anwenders ab, der einerseits hinreichend scharf umrissene, andererseits hinreichend umfassende Anfragen formulieren muss.

Metadaten sind dagegen sehr viel strukturierter: In einem Feld zum Entstehungszeitpunkt des Textes wird zwar nicht immer ein taggenaues Datum stehen; häufig wird nur bekannt sein, dass der Inhalt z. B. zwischen 1625 und 1650 verfasst wurde. Solche Informationen werden in den Metadaten typischerweise aber so formalisiert hinterlegt, dass eine automatisierte Auswertung möglich ist. In unserem Beispiel kann eine Retrieval Engine also feststellen, dass der gesuchte Zeitraum 1618-1648 mit dem vermuteten Entstehungszeitraum des Dokuments überlappt und das Dokument folglich einschlägig ist. Die Größe des Überlapps kann sogar als leicht berechenbarer Index für die Güte dieses Treffers herangezogen werden.

Ähnliche Überlegungen gelten für andere übliche Metadaten: Schlagworte werden üblicherweise aus bekannten Schlagwortlisten oder -bäumen gewählt; wenn der Name des Urhebers in mehreren Schreibweisen oder unter diversen Aliassen überliefert ist, dann enthalten die Autorenangaben typischerweise (auch) eine vereinheitlichte Personenangabe (z. B. durch Verweis auf einen Normdatenkatalog) usw.

An dieser Stelle ist die Frage nebensächlich, ob die Metadaten als Teil des Dokuments (etwa in einem TEI-Header) oder separat hinterlegt sind. Dieses Detail hat sicherlich erhebliche Auswirkungen auf die technische Realisierung einer Suche, doch aus Anwendersicht ist es unerheblich.

2.1.2 Suche in Freitext vs. ausgezeichneten Texten

Wenn der „Inhalt“ eines Dokumentes – ohne jede weitere Qualifikation – durchsucht werden soll, so wird man darunter in der Regel den vom Autor verfassten Text verstehen, den „Plain Text“. Dies ist auch die einzige Suche in Textdokumenten, bei der wir a priori immer davon ausgehen können, dass sie möglich ist. Nicht jeder in TextGrid hinterlegte Text wird notwendig von einem Editor mit Zusatzinformationen angereichert worden sein; beispielsweise kann sich eine Suche auch auf das Transkript erstrecken, das erst noch von einem Editor bearbeitet werden muss.

Wie jeder Anwender der gängigen Internetsuchmaschinen aus eigener Erfahrung bestätigen kann, erlaubt auch die Recherche im Plain Text bei geschickter Kombination der Suchwörter eine in vielen Fällen effiziente Suche. Dies gilt erst recht, wenn die Suchmaschine reguläre Ausdrücke und aussagenlogische Verknüpfungen von Suchkriterien unterstützt, so dass man sehr kompakt Schreibvarianten oder – eingeschränkt – den Kontext einer Belegstelle bei der Suche berücksichtigen kann.

Wenn ein Text mit Auszeichnungen und Zusatzinformationen angereichert wurde, dann wird man diese Informationen bei einer Suche selbstverständlich ausnutzen wollen. Recherchiert der Anwender beispielsweise Wörterbucheinträge aus verschiedenen Epochen zum Thema „Buch“, dann wird die Mehrzahl der Treffer einer Plain Text Suche auf Vorkommen dieses Wortes im Innern eines Artikels

verweisen, selbst wenn die Suche mittels der Metadaten auf Dokumente beschränkt wurde, die Wörterbücher enthalten; der Anwender muss die Treffer, in denen „Buch“ tatsächlich als Schlagwort benutzt wird, selbst aus dem Rechercheergebnis herausfiltern. Andererseits kann man davon ausgehen, dass bei der Edition eines Wörterbuches die Stichwörter als solche gekennzeichnet werden; wenn das System die Auszeichnung auswerten kann, so stehen dem Anwender viel mächtigere Abfragen zur Verfügung; es ist dann möglich, direkt nach allen Artikel zum Schlagwort „Buch“ zu suchen.

TextGrid geht davon aus, dass die Auszeichnung der eingestellten Dokumente nach einem XML-Schema erfolgt. Somit wird es möglich sein, nach und in beliebigen Auszeichnungen z. B. mit geeigneten XQueries zu suchen. Für bestimmte, wiederkehrende Abfragen lassen sich Suchmasken implementieren, welche die Details der XQuery-Syntax vor dem Anwender verstecken. Gleichzeitig werden erfahrene Anwender aber die vollständigen Möglichkeiten von XQuery ausnutzen wollen, um komplexe und von der Suchmaske nicht unterstützte Recherchen zu ermöglichen.

Die Suche unter Ausnutzung der XML-Auszeichnung bringt aber ein erhebliches Problem mit sich: Es ist nicht realistisch anzunehmen, dass TextGrid für alle (wichtigen) Textsorten abschließende Auszeichnungsschemata definieren kann, die im Sinne von Best Practice Auszeichnungen den Anforderungen aller oder auch nur der meisten Editionsprojekte genügen können. Es wird immer editionspezifische Elemente geben, die entweder dem Inhalt des edierten Werkes – beispielsweise fachspezifische Vermerke in einem historischen Wörterbuch der Winzersprache – oder der fachlichen Zielsetzung des Editors geschuldet sind. Wenn die in TextGrid eingestellten Dokumente entsprechend unterschiedlicher Schemata ausgezeichnet sind, dann wird eine spezifische XQuery aber in aller Regel nur auf einen kleinen Teil des TextGrid-Datenbestandes anwendbar sein. Mit anderen Worten: Um die Auszeichnungen in die Suche einbeziehen zu können, muss ein Anwender schon im Vorhinein wissen, nach welchem Schema die potentiellen Treffer der Suche ausgezeichnet sind. Dies widerspricht dem Anspruch, dass TextGrid eine mächtige Recherche über alle eingestellten bzw. über Archive angebundenen Dokumente ermöglichen soll.

Deshalb ist es notwendig, dass TextGrid für die wichtigsten Textsorten die typischen, in der Mehrzahl der Editionen vorkommenden Auszeichnungen und Abfragen ermittelt. Darauf aufbauend kann dann jeweils ein TEI-„Kernschema“ definiert werden. Die Nutzer von TextGrid, die selbst Dokumente in TextGrid einstellen, müssen ihre Daten nicht notwendig nach diesen Kernschemata auszeichnen; denn diese Schemata werden häufig einerseits Auszeichnungen vorsehen, die für ein konkretes Projekt nicht einschlägig sind, andererseits spezialisierte Auszeichnungen vermissen lassen. Aber die Editoren werden gehalten, eine Abbildung ihres Schemas auf das jeweilige Kernschema zu definieren. Diese Abbildung wird die spezialisierten Auszeichnungen verwerfen müssen und sie kann Elemente, die im Kernschema vorgesehen sind, aber im spezifischen Schema kein Äquivalent haben, nicht produzieren können. Aber sie ermöglicht zumindest, die Recherche in den Auszeichnungen auf Basis der Kernschemata zu formulieren und über alle ausgezeichneten Dokumente in TextGrid zu erstrecken.

In der Konsequenz muss TextGrid auch die Werkzeuge bereitstellen, mit denen diese Abbildungen programmatisch realisiert werden können, ohne dass die Anwender deshalb eigene Services in der TextGrid-Plattform installieren oder gar selbst bereitstellen müssen. Es liegt nahe, für diesen Zweck einen der in TextGrid ohnehin vorhandenen Streamingeditoren, z. B. einen XSLT-Prozessor, einzusetzen. Die Anwender können dann die geforderten Abbildungen definieren, indem sie eine Konfiguration dieses Editors, also etwa ein XSLT-Skript, hinterlegen.

Es steht den Nutzern bei der Recherche dann frei, ob sie sich bei ihrer Anfrage auf ein spezialisiertes Schema beziehen oder ob sie die XQuery entsprechend dem Kernschema formulieren. In letzterem Fall obliegt es der TextGrid-Middleware vor der Anwendung der XQuery auf ein Dokument, dieses wie vom Editor definiert auf das Kernschema abzubilden.

2.1.3 Nutzerschnittstelle

Die Nutzerschnittstelle dient der Kommunikation zwischen dem Anwender und dem Recherchesystem. Ein klassischer Anwender, der Informationen zu einem bestimmten Thema finden möchte (inhaltsorientierte Suche), hat jedoch in vielerlei Hinsicht andere Anforderungen an das User Interface als der TextGrid-Nutzer. In dem ersten Fall steht der Nutzer vor einem Suchparadoxon (etwas beschreiben zu müssen, was man nicht weiß, damit man es finden kann) und insbesondere IR-

Systeme der dritten Generation unterstützen den Anwender dahingehend, ein gutes Verständnis für den Suchraum zu bekommen, damit er seine Suchanfrage entsprechen formulieren kann. Der TextGrid-Nutzer dagegen sucht in der Regel nach Textbelegen für linguistische oder philologische Fragestellungen. In vielen Fällen benötigt er eine exakte Suche nach spezifischen sprachlichen Ausdrücken. Standardtechniken des Stringabgleichs wie *stemming* und *fuzzy matching* sind für den TextGrid Anwender nur die zweite Wahl, da eine *morphologische Analyse* die *Precision* (der Anteil der tatsächlich relevanten unter den gelieferten Treffern) und den *Recall* (der Anteil der gelieferten unter den in der Datenbasis vorhandenen relevanten Treffern) wesentlich erhöhen kann. Im Idealfall kann er aus vielfältigen Methoden wählen, um seine Suchanfrage zu erweitern oder zu verfeinern, darunter z.B. die Phrasensuche, um Kollokationen oder idiomatische Ausdrücke zu finden. In den kommerziellen Suchsystemen kann eine Phrase wie „To be or not to be“ allerdings gar nicht erst gefunden werden, da sie zu viele Stopwörter enthält.

Da präzise Suchanfragen nur formuliert werden können, wenn die Query Language entsprechend mächtig ist, d.h. reguläre und boolesche Ausdrücke umfasst, sollte diese Option für den TextGrid-Nutzer vorhanden sein. Für erfahrene Anwendern bietet sich als Anfragesprache XML Query an, die als W3C Standard auf XML Datenbanken operiert. Allerdings werden solche Anfragen schnell komplex und unübersichtlich, so dass für den ungeübten Anwender alternative Suchmasken zur Verfügung stehen sollten. Dies erfordert zudem die Kenntnis der strukturellen Eigenschaften der Dokumente, die in der Schemadefinition verankert ist. Beim Retrieval von XML-Dokumenten ist es daher oft vorteilhaft für den Benutzer, wenn er sich seine Anfrage mithilfe von Pull-down Menus zusammenklicken kann. Eine linguistische Anfragesprache wie die „Corpus Query Language“ [CSK94, CSK99], die sich bereits als Standardsprache für morpho-syntaktisch annotierte Korpora etabliert hat, kommt auch ohne komplexe Klammersausdrücke aus. Ebenso die Suche mit Sara [Bur95], die dank einer maskenbasierten Eingabemaske sehr anwenderfreundlich ist. Für Treebanks dagegen bieten sich grafikbasierte Oberflächen wie NetGraph [MOP02] an. Die TextGrid-Nutzerschnittstelle stellt daher je nach Art der Auszeichnung ein passendes, XQuery-basiertes Front-end bereit, ohne dabei die Ausdrucksfähigkeit der Anfragesprache einzuschränken. Die prominente Suchmaske von Google – hier wird eine UND/ODER-Verknüpfung bei mehreren Suchtermen impliziert – ist allerdings nur bedingt für TextGrid sinnvoll einsetzbar.

Oftmals sind mehrere Interaktionsschritte notwendig, bis der Anwender sich den Zugang zu dem gewünschten Wissen verschafft hat [SBC98]. Ein einfacher Suchprozess besteht darin, dass der Anwender eine Suchanfrage formuliert und an das System schickt. Dieses gibt dem Anwender einen Überblick über die Ergebnismenge. Schließlich wählt er ein Objekt aus oder verfeinert seine Suche (Query Refinement). Der klassische Suchprozess besteht aus durchschnittlich 10 Iterationen von 4-10 Minuten Dauer. Das Endergebnis ist eine gerankte Trefferliste – allerdings interessiert sich der Anwender in den meisten Fällen nur für die 10 höchstgerankten Dokumente (siehe [Wil06]). Der gesamte Suchprozess kann durch Mechanismen wie *Relevance feedback* optimiert werden. Er basiert auf der Idee, die häufigsten Terme eines Trefferdokuments zur Verfeinerung der Suche hinzuzuziehen und wird oft per default angewandt, um der geringen Zahl der Suchterme (in Google durchschnittlich 1,4 Terme) entgegenzuwirken, d.h. mehr Kontext zur Verfügung zu stellen. Eine solche Funktionalität wie das *Relevance Feedback* ist allerdings nur bei inhaltsbezogenen Suchen sinnvoll und muss daher nach Belieben (de)aktivierbar sein.

Auch in TextGrid sind Techniken zur Verfeinerung der Suche wichtig. Interessanterweise entsprechen gerade qualitative Korpusanalysen einem Suchprozess, in dem die Suchanfrage als Filter fungiert. Aus einem Korpus mit einer anfänglich umfangreichen Menge an Belegzitate werden stets kleinere Subkorpora gebildet. Dies geschieht in der Regel halbautomatisch, daher durch schrittweises, manuelles Herausfiltern aller falschen Treffer (false positives) aus den Suchresultaten. In diesem Zusammenhang ist es wichtig, dass auch der Parameter *Recall/Precision* vom Nutzer frei wählbar ist. Ist die Zahl der Belegzitate niedrig, so wird der TextGrid-Nutzer beispielsweise die Einstellung hoher Recall/niedrige Precision wählen, um möglichst keine relevanten Belege zu verpassen.

Das manuelle Filtern wird zum Teil darin bestehen, vorhandene Ambiguitäten im Text aufzulösen. Ist das Korpus jedoch linguistisch annotiert, entfällt diese Arbeit. Sucht der Nutzer beispielsweise ein Homonym (z.B. Bank), so kann das System anhand der lexikalisch-semantischen Annotationen (z.B. Wordnet) eine Checkbox zur Verfeinerung der Suche vorschalten (Bank-Finanz vs. Bank-Möbel).

Eine ähnliche Funktionalität bietet die *Query Expansion*. Bei einigen Anfragen ist es sinnvoll, auch Synonyme und Varianten mit in die Suche einzubeziehen. Um zu vermeiden, dass bei der automatischen Anfragerweiterung kontextirrelevante Terme hinzugefügt werden, sollte jedoch eine Rückbestätigung durch den Benutzer angefordert werden.

Eine wesentliche Stärke von TextGrid-Dokumenten ist die Metadateninformation. Sie kann entweder zur Navigation im Objektraum oder als Objektfilter genutzt werden. Dazu verwendet der Anwender entweder eine formularbasierte Suchmaske, z.B. um die Suche auf einen bestimmten Autor oder ein bestimmtes Genre einzuschränken. Alternativ kann er im Themennetz auf den erlaubten Relationen navigieren, bis hin zur Ebene der Objektdokumente.

Auch wenn es sich in TextGrid um eine föderierte Suche auf heterogenen Texten handelt, die Teil einer Dokumentensammlung sind und zu denen Querbeziehungen existieren (Inklusion, Übersetzung, etc.), sollten die unterschiedlichen Datenbanken dem Anwender weitgehend verborgen bleiben. Aswani et. al. (siehe [ATBC05]) definieren mit ANNIC ein einheitliches Nutzerinterface für die Suche auf Plain Text, XML-Text, oder RDF-Texten.

Die Nutzerschnittstelle bietet nicht nur Masken, Einstellungen, etc. für Suchanfragen sondern auch für die Präsentation der Suchergebnisse und die Indexierung. Da sich auch hier die TextGrid-Suche erheblich von den inhaltsbezogenen Suchen unterscheidet, muss die Retrieval Engine anpassbar sein.

Indexierung. Zunächst – bevor eine Suche möglich ist – wird das Recherchesystem dem Ersteller eines Dokumentes eine Funktion anbieten, dieses zu indexieren. Hier gibt es verschiedene Optionen, je nachdem, welche Teile des Textes indexiert werden sollen (z.B. nur Überschriften) und mit welchen Parametern (Sprache, Granularität, etc.). Die Indexierung kann nur die textuellen Daten und Metadaten umfassen oder wird auf bestimmte (auch mehrere) Schichten von Annotationen ausgedehnt. Während im klassischen Information Retrieval diverse Filter (Normalisierungsfunktionen und Stoppwortlisten) für eine Reduzierung der Zahl der Indexwörter verwendet werden können, ist für die meisten TextGrid-Suchen jedoch ein vollständiger Index nötig, der zudem die genauen Textpositionen beinhalten muss, damit eine Phrasensuche realisiert werden kann.

Eine automatische Normalisierung der Indexdaten z.B. in Bezug auf Groß-/Kleinschreibung ist in vielen Fällen nicht erwünscht und würde insbesondere quantitative linguistische Analyseergebnisse verfälschen [BLE06]. Stoppwortlisten müssen vom Nutzer frei wählbar sein. Auch der Tokenizer, der zur Indexgenerierung eingesetzt wird, soll vom Benutzer ausgewählt werden können.

Ergebnisanzeige. Die Anzeige der Treffer erfolgt in den meisten IR-Systemen in Form von kleinen Textauszügen (Snippets) mit Verweisen auf die Trefferdokumente. Während Lexikographen sich beispielsweise oft nur für die Textbelege interessieren – die beliebteste Darstellungsform ist die KeyWord-in-Context (KWIC) Anzeige mit alphabetischer Sortierung vor oder nach dem Keyword – bevorzugen Philologen eine strukturierte Angabe der Metadaten mit entsprechenden Verweisen auf das Text- und Bildmaterial. Mögliche Ausgabeoptionen, die benutzergerecht eingestellt werden können, sind zum Beispiel die Größe des Kontextes der KWIC-Anzeige, und die Anzeige linguistischer Annotationen in Kombination mit dem Text (farblichen Markierungen, tabellarische Anzeige).

Für den lexikographischen Einsatz ist neben einer alphabetischen Sortierung oft auch eine thematische Gliederung wichtig, insbesondere wenn sehr viele Belegstellen existieren. Hier kann ggf. ein vorberechnetes Clustering (AP5) genutzt werden, um die Suchergebnisse übersichtlicher darzustellen.

Der klassische TF-IDF Maßstab (*term frequency–inverse document frequency*), der häufig als Rankingfunktion im Text Retrieval verwendet wird, und ein Maßstab für die Relevanz eines Dokuments in Bezug auf die Suchterme darstellt, ist für TextGrid-Suchen dagegen nicht adäquat. Auch eine Bewertung der Treffer mit Verfahren wie Page Ranking ist nur mit Einschränkungen sinnvoll.

Kommentar [-tv1]: Rücksprache mit den Trierern, was da überhaupt gemacht wird und wie das technisch läuft, um den Aufwand für das Recherchetool abzuschätzen!

2.2 Text Retrieval aus technischer / technologischer Sicht

Die effiziente Suche in Textdaten ist nach wie vor Gegenstand aktueller Forschung¹. Zwar gibt es Produkte (z. T. auch open source), die den Ansprüchen von einzelnen Archiven oder Enterprise Content Management Systemen vollauf genügen. Aber es ist keineswegs klar, dass ein System „von der Stange“ die sehr vielseitigen Anforderungen von TextGrid erfüllen kann: Die Zielgruppe von TextGrid erwartet, sowohl in unstrukturierten als auch in strukturierten Texten suchen zu können (d. h. Suche in Plain Text vs. die Suche in XML-Daten); auch Metadaten müssen in die Suche eingehen können. Die verwendeten XML-Schemata sind weitgehend beliebig und müssen ggf. auch noch auf ein jeweils geeignetes Kernschema abgebildet werden. Schließlich muss TextGrid auch auf die Suche in ungewöhnlich großen Datenbeständen vorbereitet sein, wobei ein Teil der Daten nicht in TextGrid selbst liegt, sondern die Informationen von angebundenen Archiven bereitgestellt werden.

All dies legt nahe, dass TextGrid auf eine Kombination von Technologien (relationale Datenbanken, Systeme zur Speicherung semantischer Informationen wie RDF oder Topic Maps, XML-Datenbanken, Volltextindizierer etc.) setzen muss, um Textwissenschaftlern mit effizienten und mächtigen Recherchewerkzeugen bei ihrer Forschung zu unterstützen.

Wir betrachten in den nachstehenden Abschnitte einschlägige Technologien sowie eine Auswahl von konkreten Produkten, die potentiell geeignet erscheinen, Aspekte des Anforderungsspektrums an TextGrid abdecken zu können.

2.2.1 Werkzeuge zur Speicherung optimal durchsuchbarer Daten

2.2.1.1 Relationale Datenbanken

Relationale Datenbanken sind fest etablierte und probate Werkzeuge zur Verwaltung, Manipulation und Abfrage großer Datenmengen, die von Hause aus mit Mitteln zur Datenrettung- und wiederherstellung, Konsistenzprüfung, Protokollierung, Indexierung, Mehrbenutzerbetrieb und Zugriffsberechtigung ausgestattet sind. Grundvoraussetzung für relationale Datenbanksysteme ist die Anordnung der Daten in der Datenbasis in sogenannten „Relationen“ (z. B. *zugverbindung(Hamburg, Mannheim, ICE)*). Die Daten werden damit tabellenartig gespeichert und können über eine Reihe von Operationen modifiziert werden. Die einheitliche Sprache für Anfrage, Ausgabe-Formatierung und Datenerweiterung ist SQL, das in mehreren Redaktionen standardisiert worden ist (jüngst *SQL-2003*).

Traditionell werden relationale DB-Systeme als Client-Server-Systeme konzipiert (z. B. DB/2, Oracle, PostgreSQL), die über einen Administrator verwaltet werden müssen. Es gibt aber auch anwendungsinterne Systeme wie die ebenfalls von Oracle vertriebene Berkeley-DB.

Eine Weiterentwicklung der relationalen Datenbank-Systeme sind die objektrelationalen Datenbank-Systeme, die Mengen von Objekten in Relation zueinander setzen können und auch eine Unterstützung für nicht-interpretierbare Daten und große Objekte wie Binärdateien mit Multimedia-Inhalten mitbringen. Im TextGrid-Einsatz wäre für solche Datenbank-Systeme die Verwaltung der verschiedensten Dokumente (Grafiken, Korpora, Projekt-Dokumente) mitsamt ihren Metadaten denkbar.

2.3 XML-Datenbanken

XML-Datenbanken bieten die gleichen Vorteile wie relationale Datenbanksysteme bzgl. des Daten-Managements, basieren jedoch auf XML-Dokumenten bzw. deren Inhalten, die gespeichert, manipuliert und abgefragt werden können. Dabei lassen sich zwei große Gruppen der XML-Datenbanken unterscheiden: XML-fähige („XML-enabled“) und XML-native Datenbank-Systeme.

Die XML-fähigen Datenbank-Systeme arbeiten intern meist nach dem relationalen Datenmodell, auf das beim Abspeichern die Daten einer XML-Datei abgebildet werden. Aus dem Ergebnis der Anfrage

¹ Dieses Thema ist sogar nach wie vor Gegenstand regelmäßiger Konferenzen, etwa der jährlichen Text Retrieval Conference (TREC), <http://trec.nist.gov/>.

wird dann eine neue XML-Datei generiert. Die ursprüngliche Struktur des XML-Dokumentes geht somit verloren. Dazu kommt ein bisweilen nicht unerheblicher Programmieraufwand, die Strukturen einer XML-Datei auf ein relationales Modell abzubilden, wenn die Struktur rekursive Elemente, Inhalte gemischten Typs, etc. aufweist.

Anders bei XML-nativen Datenbank-Systemen, die intern mit einem XML-Datenmodell arbeiten, d.h. die Struktur der zu speichernden XML-Datei in ein XML-Baummodell wie SAX, DOM oder BTree abbilden. Darum kommt auch nicht SQL als Anfragesprache zum Einsatz, sondern eine Reihe von sich noch in der Standardisierungsphase befindender Sprachen wie XQuery oder XPath (für Anfrage) und XUpdate (wie Datenmanipulation). Dafür sind sie weniger zuverlässig als relationale Datenbanksysteme und haben weniger Mittel zur Verfügung, um Datenintegrität zu gewährleisten. Aber daran wird gearbeitet.

Bei XML-nativen Datenbank-Systemen sind die drei am weitesten verbreiteten Indexierungs-Strategien a) die Werte-Indexierung, bei der Inhalte der XML-Elemente und die Attribut-Werte indiziert werden, b) die Volltext-Indizierung (näheres siehe unten) und c) die Struktur-Indexierung, bei der Teil-Bäume aus dem XML-Baummodell indiziert werden.

Die Probleme bei XML-nativen Datenbank-Systemen bestehen darin, dass sie bei großen Datenmengen noch wenig performant sind, eine relativ junge Technologie darstellen und es noch an Spezialisten für ihren Einsatz mangelt. Dennoch können sie gerade für die Indexierung und von Verwaltung von Daten, die in den TEI-Dokumenten in TextGrid gespeichert sind (z. B. Autorennamen, Werkstitel, etc.) von großem Nutzen sein. Solche Informationen können jedoch auch leicht extrahiert werden und dann mit XML-fähigen Datenbanken verwaltet werden. Die XPath- und XQuery-Unterstützung für die etablierten relationalen Datenbank-Systeme wie PostgreSQL ist in Arbeit. Der Einsatz XML-fähiger Datenbanken macht dann Sinn, wenn das zugrundeliegende relationale Datenbank-System noch für andere Zwecke (z. B. Datei-Verwaltung) eingesetzt werden soll. Dafür sind sie aber nicht wie XML-native Datenbank-Systeme unabhängig vom verwendeten XML-Schema. Ändert sich dieses, muss auch das Schema der relationalen Datenbank angepasst werden.

Open-Source-Kandidaten für XML-fähige relationale Datenbanken sind MonetDB/XQuery und PostgreSQL, wobei der XML-Support bei letzterem noch in der Entwicklung ist. Als native XML-Datenbanken sind im Open Source Bereich eXist, Berkeley DB-XML, Xindice, Sedna und Timber zu nennen.

2.3.1.1 Volltextdatenbanken bzw. Volltext-Indizierer

Volltextdatenbanken sind im Grunde genommen Systeme, die lediglich eine Volltext-Indizierung vornehmen und einen effizienten Zugriff auf größtmögliche Zeichenfolgen erlauben, wie es man es von den Google-Suchanfragen her kennt.

Volltext-Indizierung kann einmal durch reine Indizierer erfolgen wie sie von Suchmaschinen im Internet oder vom heimischen PC-Desktop her bekannt sind, sowie innerhalb von XML-Datenbank-Systemen, bei denen so der Zugriff auf die reinen Dokument-Daten beschleunigt wird.

Volltext-Indizierung ist dann interessant, wenn eine große Zahl von Dokumenten auf ganze Wortphrasen hin durchsucht werden muss. Mit Lucene, einer Open-Source-Bibliothek, auf der Beagle basiert und die für verschiedene Programmiersprachen zur Verfügung steht, lässt sich auf bereits etablierte Technologie leicht zurückgreifen. Dafür ist Lucene bislang aber noch völlig XML-ignorant: Beim Indexieren werden die XML-Tags wie gewöhnlicher Text behandelt und mitindexiert; dass es sich dabei um Metainformationen handelt, wird von Lucene nicht berücksichtigt.

2.3.2 Suche über Institutionsgrenzen hinweg

Eine verteilte Suchinfrastruktur kann aus diversen Gründen interessant oder notwendig sein. Existieren mehrere, dezentral und evt. unabhängig organisierte Portale, deren Inhalte in einem Suchsystem gesammelt verfügbar gemacht werden sollen, ist deren Verknüpfung durch ein verteiltes Suchsystem mitunter die einzig mögliche Herangehensweise. Nebst einer organisatorischen bzw. politischen

Motivation können sich aus technischer Sicht Vorteile ergeben, da potenziell die Verfügbarkeit und die Performanz eines Suchsystems gesteigert werden kann.

In einer verteilten Suchinfrastruktur sind verschiedene Schritte nötig, um verteilte Suchknoten anzusprechen und die Ergebnisse anschließend wieder zusammenzubringen.² In jedem dieser Schritte kann die Kohärenz der Sucharchitektur gesteigert werden, und damit die Suchergebnisse für den Benutzer verbessert werden. Neben den im Folgenden präsentierten drei Schritten müssen natürlich innerhalb eines Suchkonsortiums die rechtlichen Voraussetzungen und Zugriffsbedingungen abgeklärt werden; gemeinsame Softwareschnittstellen geschaffen werden; und die einzelnen Suchknoten auch verlässlich verfügbar sein (nach entsprechenden Policies).

1) **Durchkämmen der Suchknoten -**

Bei der Extraktion relevanter Information aus der Datenbasis (Crawling) ist ein gemeinsames Verständnis über die Art der Daten und die Möglichkeiten zur Suche in allen Knoten eines Suchkonsortiums notwendig. Für eine Metadatenuche ist natürlich Voraussetzung, dass die Metadatenschemata in den Teilen übereinstimmen, wo Suche möglich sein soll. Genauso ist es wichtig, dass ein gemeinsames Verständnis über die Inhaltsdaten und was durchsucht werden soll in einem Konsortium vorliegt. Z.B. kann nicht über alle Daten im Suchraum eine bestimmte TEI Annotation durchsucht werden, wenn nicht der gesamte Suchraum aus TEI Dateien besteht, oder wenn die Schemata nicht überlappen. Zur Homogenisierung des Suchraums können Standards vorgegeben werden, unterschiedliche Datentypen aufeinander abgebildet werden, oder der Benutzer bekommt eine Suchmöglichkeit nicht zur Verfügung gestellt.

2) **Aufbau der Suchindices -**

Während des Durchkämmens der Datenbasis extrahiert der Suchagent (Crawler) eine Reihe von Daten in einen Index, die zur Beantwortung einer Suchanfrage hinzugezogen werden. Um die Indices zwischen den verschiedenen Knoten vergleichbar zu machen, müssen die ihnen zugrunde liegenden Modelle übereinstimmen. Das am weitesten verbreitete Modell für Suche ist das Vektorraummodell, aber auch das Boolesche Modell oder probabilistische Modelle werden angewendet. [Lew05] Weiters sollte natürlich gesichert sein, dass die Daten, die im Suchindex registriert werden, auch zwischen den Knoten dieselben sind. So könnten z.B. Stoppwortlisten unterschiedliche Bindewörter filtern oder manche Knoten könnten Algorithmen zur Stammformenextraktion vorschalten.

Aufbau der Statistiken: Indizierung: Stopplisten, etc. Auch einheitliche Identifier für die untersuchten Objekte (z.B. normalisierte URLs, ein Satz charakterisierender Metadaten) sind an dieser Stelle wichtig zur Zusammenführung verschiedener Suchlisten im nächsten Schritt.

3) **Zusammenführen der Suchergebnisse -**

Nachdem eine Suchanfrage an verteilte Suchindices ausgesandt wurde, erhält man eine Reihe von Ergebnislisten. Dem Benutzer können nun alle diese Ergebnislisten separat als die verschiedenen Antworten auf seine Anfrage präsentiert werden. Alternativ könnten diese unterschiedlichen Listen in eine Liste zusammengeführt werden. Dazu müssen eventuell Ergebnisse, die in mehreren Listen vorkommen, herausgefiltert werden, um Mehrfachnennungen zu vermeiden. Die wohl größte Herausforderung beim Zusammenführen der Suchergebnisse ist allerdings ein einheitliches Ranking. Algorithmen zum Relevance Ranking wie TFxIDF sind abhängig von der Anzahl der indizierten Dokumente und können daher nicht direkt zwischen separaten Datenbeständen vergleichen. Zumal oftmals eine Suchföderation zwischen spezialisierten Archiven aufgebaut wird und die unterschiedlichen Schwerpunkte ein einheitliches Ranking stören. Der Austausch von Gewichtungsfaktoren zwischen allen beteiligten Datenbanken ist zu einem homogenen Relevance Ranking daher notwendig. [CFK⁺06]

Drei grundlegende Architekturtypen zum Aufbau einer Suchföderation sind: der Aufbau eines zentralen Index, Metasuche und föderierte Suche.

² Obs. in der Literatur gibt es unterschiedliche Ansätze zur Terminologie, Unterteilung in Einzelaufgaben, etc., je nachdem aus welcher fachlichen Sicht und mit welchen Anforderungen an die Sache herangegangen wurde. Die Unterteilung hier ist inhaltlich an Vorbilder (z.B. [CFK⁺06]) angelehnt.

2.3.2.1 Zentraler Index

Für eine Suche über verteilte Datenquellen ist die Erstellung eines zentralen Index eine pragmatische Herangehensweise. Für eine reine Metadatensuche bieten sich OAI-PMH basierte Crawler an - z.B. der ARC Harvester³ -, und auch für Volltexte gibt es bereits traditionelle Lösungsansätze (z.B. Lucene), wie auf Seite 10 beschrieben. Mit einem zentralen, homogenen Index treten die oben beschriebenen Schritte 2 und 3 nicht ein. Auch bei einem zentralen Index ist allerdings die Schaffung von Standards im Bereich der Metadaten und Datenmodelle vorteilhaft, um möglichst homogene Ergebnisse zu erlangen.

2.3.2.2 Metasuche

Eine Metasuche ist die lose Kopplung einzelner Suchportale, die untereinander nicht verknüpft sind. Gemeinsame Standards können in so einer isolierten Umgebung nicht etabliert werden. Die Metasuche greift lediglich die Ergebnislisten diverser Suchmaschinen ab, und führt diese an einem Punkt zusammen. Beim Ranking gibt es eventuell heuristische Methoden, um die diversen Ergebnislisten zusammenzuführen, aber auch diesem Schritt sind Grenzen gesetzt.

Metasuche ist vor allem im Internet aufgekommen, mit Suchmaschinen wie MetaCrawler oder HotBot. Wirklich durchgesetzt hat sich die Metasuche aber nicht.

2.3.2.3 Föderierte Suche

Eine föderierte Suche wird in einer weitgehend kooperativen Umgebung installiert. Die Partner können diverse Mechanismen setzen, um eine möglichst homogene Suche aufzubauen. Je nach Art des Suchraums und Anforderungen an die Suche können durch gemeinsame Standards und Modelle die Schritte 1 und 2 homogenisiert werden. Dazu können die einzelnen Knoten auch unterschiedliche Suchtechnologien einsetzen, solange die abgesprochenen Standards darin umgesetzt werden können. Einzig ein einheitliches Ranking in Schritt 3 stellt derzeit noch technische Herausforderungen, wie oben beschrieben. Mit der in [CFK⁺06] beschriebenen Vorgehensweise für das Suchportal *vascoda* - einem Zusammenschluss aus diversen Bibliotheken, Fachinformationszentren und anderen Partnern - kann über die Installation zusätzlicher Protokolle ein dynamischer Austausch der notwendigen Gewichtungsfaktoren erfolgen. Damit kann auch in einer föderierten Suche eine Suchqualität vergleichbar einer zentralen Suche installiert werden.

Internationale Firmen haben oft verteilte, heterogene Datenbanken in ihren diversen Niederlassungen und Abteilungen, die sie zur besseren Nutzung des firmeninternen Wissens verknüpfen wollen. Es gibt daher schon diverse kommerzielle Anbieter - darunter Inktomi/Veritas, EMC, und andere -, die föderierte Suche anbieten, und lokale Systeme mittels proprietären Wrappern oder eigenen Schnittstellen anbinden. Open Source Suchsysteme mit Unterstützung für föderierte Suche gibt es derzeit noch wenige. Neben Ansätzen für verteilte Suche über das Z39.50/ZING Protokoll könnten aber die Aktivitäten in *vascoda* [CFK⁺06] und dem SDARTS Protokoll einen zukünftigen Trend vorzeichnen.

2.3.2.4 Vorgeschaltete Filter bei verteilten Suchen

Mit Blick auf die Performance haben verteilte Suchen zwangsläufig Nachteile: Die Zeit, die zum Abschluss einer verteilten Suche verstreicht, wird von dem Server bestimmt, dessen Antwort am längsten auf sich warten lässt. Wenn ein einzelner Server nur mit der Wahrscheinlichkeit q länger als von den meisten Anwendern toleriert für eine Antwort benötigt, so steigt diese Wahrscheinlichkeit bei der Verteilung einer Anfrage auf n Server (vereinfacht) auf $1 - (1-q)^n$. Mit anderen Worten, die Wahrscheinlichkeit, dass das Endergebnis einer Recherche unakzeptabel lange auf sich warten lässt, wächst exponentiell in der Anzahl der Server, die befragt werden muss.

³ ARC - OAI Federated Search Harvester, <http://arc.cs.ou.edu>

Außerdem kann das in Schritt 3 auf Seite 11 beschriebene Zusammenführen der Antworten der einzelnen Server je nach Art der Recherche und der zu Grunde liegenden Datenbank sehr aufwändig sein, weil hierfür nicht die bei einem zentralistischen Ansatz vom Datenbanksystem vorberechneten Indizes und Zugriffstabellen zur Verfügung stehen.

Beide Überlegungen zeigen, dass auch bei verteilten Recherchen die Anzahl der parallel befragten Server möglichst minimiert werden sollte. Es kann sich deshalb lohnen, vor der eigentlichen Recherche einen Filter vorzuschalten, der Server, von denen vorhergesagt werden kann, dass sie keine Treffer liefern werden, aus der zu befragenden Liste herausnimmt. Eine Filter-Technik, die hierfür interessant erscheint, sind so genannte *Bloom-Filter* [Blo70,Mit01].

Bloom-Filter setzen voraus, dass eine endliche Liste von Schlagwörtern oder häufigen Suchbegriffen existiert. (Zu dieser Liste können allerdings jederzeit und mit geringem Aufwand Elemente hinzugefügt werden.) In einer Vorabrechnung wird für jeden derartigen Suchbegriff und jeden Server bestimmt, ob der Server für diesen Begriff einen Treffer enthält. Das Ergebnis wird in einer Tabelle hinterlegt, die verlustbehaftet komprimiert ist. Der Verlust äußert sich aber ausschließlich derart, dass in der Tabelle *falsch positive* Ergebnisse auftauchen können. D. h., es hat möglicherweise den Anschein, als würde ein bestimmter Server zu einem gegebenen Suchbegriff einen Treffer liefern, obwohl dies nicht zutrifft. Falls der Server tatsächlich einen Treffer melden wird, dann wird dies immer korrekt in der Tabelle vermerkt.

Wenn die verteilte Recherche einen der im Bloom-Filter berücksichtigten Suchbegriffe einschließt, so kann über den Bloom-Filter eine Menge von Servern bestimmt werden, die potentiell einen Treffer für diesen Suchbegriff bieten. Im Idealfall ist diese Menge erheblich kleiner als die Menge aller zur Verfügung stehenden Server, so dass die Recherche signifikant beschleunigt werden kann.

Falls die Anfrage keinen der im Bloom-Filter vorgesehenen Suchbegriffe umfasst, dann muss der Filter übersprungen werden. In diesem Fall wird die Recherche aber auch nicht wesentlich verzögert, so dass durch den Bloom-Filter kein Nachteil entsteht. Weil die Liste der Suchbegriffe jederzeit ergänzt werden kann, lässt sich der Bloom-Filter auch zur Laufzeit optimieren.

2.3.2.5 TextGrid Sicht

TextGrid plant eine verteilte Suche über angebundene Textarchive. Wie in der *TextGrid Architektur* besprochen, gibt es verschiedene Integrationsstufen für angebundene Textarchive. Derzeit ist zwar eine Metadatensuche in allen Integrationsstufen, eine Volltext und Strukturdatensuche allerdings nur in der höchsten Integrationsstufe - dem Data Grid geplant. Da das Data Grid an sich schon die verteilten Daten virtualisiert, bietet sich die Installation eines zentralen Index an. Dies ermöglicht eine homogene Suche bei minimalem Aufwand für die anzubindenden Textarchive. In einem zukünftigen Schritt könnte überlegt werden, ob in der zweiten Integrationsstufe auch eine föderierte Suche eingebettet werden könnte. Diesbezügliche Entwicklungen wie z.B. die Erfahrungen im Rahmen von *vascoda* würden hierzu herangezogen.

2.3.3 Text Mining Techniken

Das Gebiet ‚Text Mining‘ beschäftigt sich mit der Extraktion von Wissen aus unstrukturierten Texten. Dazu kommen Methoden aus den Bereichen Information Retrieval, Computerlinguistik, Data Mining und Maschinelles Lernen zum Einsatz. Text-Mining-Anwendungen benötigen eine robuste Infrastruktur mit adäquaten Tools zur Dokumentenverwaltung (für unterschiedlicher Formate und mehrfach annotierte Texte) sowie NLP-Tools zum Tokenisieren, PoS-Taggen, Parsen oder Chunks, Summerizing, etc. Die bekanntesten Frameworks sind GATE (General Architecture of Text Engineering⁴) und UIMA (Unstructured Information Management Architecture⁵) von IBM. Ein wesentlicher Bestandteil von GATE ist die IE-Komponente ANNIE für die Eigennamenerkennung (Personen, Orte, Institutionen, etc.) oder Terminologieextraktion. Die Verarbeitungszeit ist auch für

⁴ <http://gate.ac.uk>

⁵ <http://www.research.ibm.com/UIMA>

große Textmengen relativ gering, da das Verfahren auf (einer Pipeline von) endlichen Automaten basiert. Text Mining Techniken werden zunehmend zur automatischen Erzeugung von Annotationen innerhalb spezifischer Fachgebiete (insbesondere den Life Science) und dem ‚semantic web‘ verwendet.

2.3.4 Semantische Suche

Unter ‚semantischer Suche‘ werden diverse Ansätze zur Erweiterung traditioneller Suchtechnologien verstanden, die inhaltliche Bezüge zwischen Suchbegriffen in den Suchprozess einbeziehen und damit die eigentliche Bedeutung einer Anfrage herausarbeiten versuchen.⁶ Für den Nutzer ergeben sich durch semantische Erweiterungen diverse Möglichkeiten. Eine einfache und nützliche Anwendung ist Googles Helferlein "Meinten Sie: 'semantic' ", wenn sich bei der Anfrage ein Tippfehler 'semantic' eingeschlichen haben sollte. Weitergehende Unterstützung würde ein Suchmechanismus bieten, der alternative Anfragen anbietet. Z.B. würde eine Suche nach "Semantik" eine Verfeinerung der Suche in verschiedene Kategorien wie "Sprachwissenschaft, Pragmatik" oder "W3C Semantic-Web" anbieten, oder umgekehrt eine Suche nach "RDF" die Erweiterung der Suchanfrage auf "Semantic-Web". Das Suchportal 'KartOO'⁷ demonstriert eindrucksvoll eine Form der visuellen Navigation in einem Suchraum. Hierbei werden auf eine Suchanfrage die semantischen Bezüge zwischen den einzelnen Treffern herausgearbeitet und in einer Karte dargestellt. Z.B. werden bei einer Suche nach "Jaguar" die Webseiten des Autoherstellers und Autofanseiten nahe bei einander dargestellt und mit den Stichwörtern "car" und "modell" versehen, während diverse Seiten zur Großkatze Jaguar an anderer Stelle in der Karte mit "habitat" und "feline" beschrieben wird. Eine Fokussierung der Suche auf eines der beiden Konzepte ist möglich.

Semantische Suche wird aktuell als der nächste evolutionäre Schritt in Suchtechnologien diskutiert. Aufgrund der Komplexität und der bereichsspezifischen Ausprägungen von semantischen Technologien sind allerdings im Open Source Bereich noch kaum produktionsreife Anwendungen verfügbar. Diverse Firmen haben sich auf das Anpassen von semantischen Suchtechnologien auf Firmenumgebungen spezialisiert. Z.B. bieten Acetic/Cyberlex mit einer Kombination aus Textanalyse (Tropes) und Suchtechnologie (Zoom) eine Technologie zur Erschließung und Kontextualisierung firmeninterner Informationsumgebungen. Caboodle Networks entwickelt die Suchmaschine Semantical⁸ als Management-Suite für Firmenportale. Die wohl größte Presse machte unlängst das deutsche Suchprojekt "Theseus", das mittels semantischer Suche eine Alternative zu 'Google' aufbauen will.⁹

Primär kann man zwei Herangehensweisen unterscheiden, um semantische Suche umzusetzen: symbolische Methoden, und statistische bzw. subsymbolische Methoden.

In **symbolischen Methoden** werden Begriffe in einem Netzwerk zu einander in Bezug gesetzt. Die entstehenden formalen Ontologien definieren die Bedeutung von Begriffen anhand ihrer Umgebung. Die Suche kann in dem Beziehungsnetzwerk erweitert werden, und Ambiguitäten von Begriffen können schnell erkannt werden.

Statistische oder subsymbolische Methoden sind nicht auf die manuelle Definition von Ontologien angewiesen, sondern analysieren ihren Suchraum und erstellen automatisch mögliche Klassifizierungen. Das oben beschriebene KartOO basiert auf diesem Prinzip; [Cil04] beschreiben,

⁶ Manchmal werden auch andere Konzepte 'semantische Suche' genannt, die hier aber nicht tiefergehend betrachtet werden. Z.B. wird in anderem Kontext die Verwendung natürlicher Sprache zur Anfrage 'semantische Suche' genannt. Eine natürlichsprachige Anfrage wäre z.B. "Wie heißt die Hauptstadt von Frankreich?", die als Antwort "Paris" geben sollte (anstatt einer Webseite mit einem Hauptstädtequiz ohne die entsprechenden Antworten).

⁷ <http://www.kartoo.com/>

⁸ Die Open Source Suchmaschine Semantical (<http://semantical.org/>) von Caboodle Networks (<http://www.caboodlenetworks.com/>).

⁹ Heise News: IT-Gipfel: Quaero heißt jetzt Theseus. 18.12.

wie Google dazu 'missbraucht' werden könnte; und selbst Multimedia Inhalte können durch Neuronale Netze auf diese Weise erschlossen werden. [MIRifs]

2.3.4.1 TextGrid Sicht

TextGrid hat durch das Wörterbücher-Netzwerk, das als Erschließungsmaterial in TextGrid eingebunden wird, bereits eine vorhandene Basis zur Erstellung formaler Ontologien. Die Wörterbücher selbst sind mehr als Wortlisten; mittels in Trier entwickelter Methoden zur statistischen Analyse können semi-automatisch Begriffsnetzwerke entwickelt werden. Da in TextGrids Wörterbuch-Netzwerk Wörterbücher aus verschiedenen Zeiten zusammenkommen, könnte die Begriffsentwicklung in dem Begriffsnetzwerk mit abgebildet werden. Die entstehende Ontologie, die inhaltliche Beziehungen und zeitliche Abhängigkeiten abbildet, wäre eine hervorragende Möglichkeit zur Erweiterung der Suche im TextGrid Recherchewerkzeug.

Während in der beschriebenen TextGrid Wörterbuch-Ontologie eine inhaltliche Beziehung zwar verzeichnet aber nicht expliziert ist, könnten weitergehend bestimmte Relationen definiert werden. Also z.B. könnten Gegenbegriffe wie "Gesundheit" und "Krankheit", Ober- und Unterbegriffe, und andere Beziehungen explizit definiert werden. Ähnliche Ansätze verfolgen auch WordNet¹⁰, GermaNet¹¹, und andere WordNet Projekte¹², die zumeist aus dem linguistischen Bereich stammen und entsprechend Beziehungen mit grammatikalischen Bezügen definieren. Während dieser Ansatz eine tiefgehende semantische Erschließung erlaubt, bedeutet er einen erheblichen manuellen Aufwand. Eine Umsetzung in TextGrid würde sich dementsprechend in einem ersten Schritt an einer "flacheren" (wie im oberen Absatz beschrieben) aber ausbaufähigen Ontologie orientieren.

2.3.5 sprachspezifische Suche

Im vorherigen Abschnitt fiel bereits das Stichwort „Textanalyse“. Unter diesen Begriff fallen verschiedene Techniken von unterschiedlicher Komplexität, die auch z. T. aufeinander aufbauen. Ihnen ist allen gemein, dass sie inhärent sprachspezifisch sind.

Die vielleicht einfachste dieser Techniken berücksichtigt die Aussprache beim Vergleich; wenn zwei Worte nahezu gleich ausgesprochen werden, dann werden sie als gleich behandelt. Für die englische Sprache liefert der bereits lange vor den ersten Computern entwickelte Soundex Algorithmus überraschend gute Ergebnisse; mittlerweile existieren noch weiter verbesserte Algorithmen, so dass ein phonetischer Vergleich im Englischen praxistauglich scheint. Auch für das Deutsche existiert mit dem so genannten Kölner Verfahren eine angepasste Variation des Soundex Algorithmus.

Ebenfalls aufs einzelne Wort bezogen sind Vergleiche, die sich auf die jeweiligen Lemmata beziehen. Dadurch werden verschiedene Flexionsformen eines Wortes als äquivalent behandelt. Weil die Flexion von Worten ebenso wie ihre Aussprache für jede Sprache verschieden ist, kann auch diese Form der unscharfen Suche nur realisiert werden, wenn ein Lemmatisierer für die jeweilige Sprache bzw. Sprachstufe implementiert ist.

Besonders komplex wird der Vergleich, wenn hierbei auch noch der Satzkontext berücksichtigt werden soll. Daraus können wertvolle Informationen für die semantische Suche gewonnen werden, weil dadurch im Idealfall Synonyme oder verschiedene Bedeutungsfelder eines Wortes disambiguiert werden können.

2.3.6 multilinguale Suche

Die TextGrid-Dienste gehen davon aus, dass alle Textdaten im Universal Character Set (Unicode, UCS) [Uni07] kodiert sind; nötigenfalls müssen die Daten beim Einlesen *on the fly* transformiert werden. Dadurch ist sichergestellt, dass TextGrid unabhängig von den Sprachen und Schriftsystemen

¹⁰ <http://wordnet.princeton.edu/>

¹¹ <http://www.sfs.uni-tuebingen.de/lzd/>

¹² <http://www.globalwordnet.org/>

ist, in denen die Korpora verfasst sind – selbst historische oder „exotische“ Schriften und ausgefallene Sonderzeichen können so in aller Regel kodiert werden. Dennoch ergeben sich bei der Suche in UCS-kodierten Korpora Probleme, die nicht vernachlässigt werden können. (Dieser Abschnitt basiert in weiten Teilen auf [Küs01].)

Eine Suche macht offensichtlich nur Sinn, wenn wir Daten vergleichen können. Dies ist bei Texten, die beispielsweise den in Mitteleuropa weit verbreiteten Zeichensatz ISO 8859-1 nutzen, trivial, weil es nur einen bitweisen Vergleich von Zeichenketten erfordert. Um in Unicode-Daten suchen zu können, muss das unter dem Schlagwort *canonical equivalence* bekannte Problem berücksichtigt werden, dass mehrere Sequenzen von Unicode Codepoints ein und dasselbe Zeichen (visuell wie semantisch) darstellen können. (Bsp: Ein kleines e mit Akut-Akzent – é – kann sowohl durch den einzelnen Codepoint U+00E9 als auch die Sequenz U+0065 U+0301 kodiert werden.) Jede dieser alternativen Kodierungen muss bei einem Vergleich als äquivalent behandelt werden, d. h. jeder String muss zunächst normalisiert werden. Das W3C Konsortium benennt in einem technischen Bericht [Dür98] eine Reihe der Schwierigkeiten, die dabei auftreten. Ein ähnliches Problem tritt auf, wenn man XML-Dateien in einen Volltextindex aufnimmt: Zeichen können in XML grundsätzlich durch korrespondierende *Entities* ersetzt werden, d. h. anstelle eines *ä* steht etwa eine der Zeichenketten `ä`, `ä` oder `ä`. Auch diese *Entities* müssen – wenn es sich tatsächlich um XML-Daten handelt – vor der Indizierung konvertiert werden.

Des weiteren erwarten Anwender, dass sie auch nach gängigen Transkriptionen suchen können: Eine Suche nach Achilleus sollte auch Belegstellen von Αχιλλεύς in einem Korpus der Homerschen Überlieferung finden (und umgekehrt), obwohl beide Zeichenketten klar verschieden sind. Der gängige Lösungsansatz für dieses Problem nutzt Normdatenkataloge (z. B. [RAK04]), obwohl diese natürlich nie allumfassend sein können.

Schließlich bleibt noch eine Anforderung an die Nutzerschnittstelle: Wenn in einer Recherche Worte vorkommen, die nicht in den Normdatenkatalogen enthalten sind, dann müssen diese Worte irgendwie in ihrer ursprünglichen Schrift in die Suchmaske eingegeben werden. Wer Zitate der alttestamentarischen Offenbarung des Gottesnamen (Ex. 3,14) sucht, muss in der Lage sein, יהוה in die Suchmaske einzugeben. Weil nicht gesagt ist, dass im Betriebssystem des Nutzers eine hebräische Tastaturbelegung installiert ist (und der Benutzer diese auch nicht notwendig beherrscht), muss das Nutzerinterface optional anbieten, Zeichen auch über einen Dialog zur Auswahl von Sonderzeichen unabhängig von der Tastatur einzugeben.

3 Auswahl existierender Werkzeuge

Im folgenden geben wir einen kurzen Überblick über eine Auswahl von Programmen und Werkzeugen, die entweder für den Aufbau der Recherchefunktionalität in TextGrid in Frage kommen oder die wir als Beispiele für existierende Recherchertools untersuchten.

3.1 eXist

Bei eXist¹³ handelt es sich um eine XML-native Datenbank aus der OpenSource-Szene, die auf Java basiert (demnach auch eine Java Runtime Environment benötigt) und auf einer Client-Server-Architektur beruht. Ihr Administration von eXist erfolgt über ein Web-Interface.

Für den Entwickler stellt eXist eine Reihe von Schnittstellen zur Verfügung: SOAP, XML:DB API, XML-RPC API. Als Anfragesprache steht XQuery zur Verfügung, mit dem sich in Verbindung mit XSLT, XHTML, CSS und JavaScript auch sehr leicht Web-Applications implementieren lassen. Die XQuery-Implementierung entspricht den derzeitigen Standardisierungs-Entwürfen mit Ausnahme als optional definierter Schema-Import- und Validierungs-Features. Ferner werden XUpdate, XPath und teilweise XInclude und XPointer unterstützt.

eXist verfügt über fünf verschiedene Indexierungs-Arten, darunter Struktur-Index und Volltext-Index und Index über XML:IDs. Weitere Indexierungs-Verfahren wie N-Gram-Indexierung sind in Arbeit.

¹³ <http://www.exist-db.org/facts.html>

Zusätzlich ist eXist mit dem Features eines numerischen Indexierungsverfahrens für XML-Knoten ausgestattet, wodurch bei der Query-Abarbeitung ein speicher-intensives Traversieren des Dokument-Baumes vermieden wird.

An eXist wird seit 2000 gearbeitet, zur Zeit liegt Version 1.1.1 vor. Gemäß der veröffentlichten Roadmap des Entwicklers ist künftig noch mit signifikanten Erweiterungen zu rechnen.

3.2 Berkeley DB-XML

Oracle Berkeley DB XML¹⁴ ist eine XML-Erweiterung für die OpenSource-Datenbank Berkeley DB. Es ist eine eingebettete Datenbank, die die Zuverlässigkeit (z. B. ACID-Transaktionen und Recovery) und Performanz von relationalen Datenbank zusammen mit XML-Funktionalität (z. B. W3C-konformes XQuery 1.0 und XPath 2.0) zur Verfügung stellt („Oracle Berkeley DB XML adds a document parser, XML indexer and XQuery engine on top of Oracle Berkeley DB to enable the fastest, most efficient retrieval of data“). Trotzdem speichert sie XML-Dokumente nativ. Eine Volltext-Indexierung wird nicht unterstützt.

3.3 Apache Xindice

Xindice¹⁵ ist eine native XML-Datenbank mit Unterstützung für XPath und XUpdate und den APIs XML:DB sowie XML-RPC. Es liegt in der Version 1.1 vor und scheint sich in stetiger Weiterentwicklung zu befinden, allerdings weit weniger ambitioniert als eXist. Es ist ebenfalls Java-basiert und basiert auf einer Client-Server-Architektur. XQuery scheint nicht unterstützt zu werden.

3.4 Apache Lucene

Lucene¹⁶ ist eine auf Java basierte Bibliothek zur Indexierung und Durchsuchung von Volltexten. Sie besticht durch ihre Effizienz und Performanz und ihre zahlreichen Portierungen in andere Sprachen (z. B. Perl, Python, C#, etc.).

Lucene erstellt einen Index, dem später weitere Texte hinzugefügt werden können. Die Query-Komponente kann aus diesem Index beliebig lange Phrasen als Suchanfragen finden und gibt die besten Dokumente als erstes aus. Dieses Feature ist mit einer der Hauptgründe, weshalb Lucene von Desktop-Suchsystemen wie Beagle oder Suchmaschinen wie Nutch eingesetzt wird.

Wie oben (Abschnitt 2.3.1.1) bereits erwähnt, ist Lucene nicht von sich aus in der Lage, XML-Daten korrekt zu verarbeiten, das Markup wird wie normaler Text behandelt.

3.5 Suchwerkzeuge in „Der junge Goethe in seiner Zeit“, Internet- und CD-ROM-Version

Beim *Jungen Goethe* [EJW98] handelt es sich um eine wissenschaftliche Edition der Werke des jungen Goethe. Neben der Buchkomponente ist auch eine CD-ROM mit allen Texten und weiteren Materialien Teil der Edition. Wir betrachten in diesem Abschnitt, wie gut sich typische literaturwissenschaftliche Fragestellungen mit dem auf der CD-ROM enthaltenen Recherchetooll (FolioViews) sowie den Werkzeugen einer online-Ausgabe des Jungen Goethe beantworten lassen.

Bei einfachen Suchen bzw. Ähnlichkeitssuchen ohne Platzhalter, mit denen etwa verschiedene Schreibungen eines Wortes gefunden werden sollen, variiert die Qualität der Suchergebnisse. Nicht gefunden wurden z. B. manche Varianten mit i/y (vielerlei / vielerley) oder mit bzw. ohne h (Gemalin / Gemahlin). Varianten werden regelmäßig nicht erkannt, wenn sie sich an mehr als einer Stellen unterscheiden oder Sonderzeichen wie einen Bindestrich enthalten. Ansonsten lässt sich kaum vorhersagen, welche Varianten gefunden werden, so dass ein Anwender keinen Anhaltspunkt hat, ob

¹⁴ <http://www.oracle.com/technology/products/berkeley-db/xml/index.html>

¹⁵ <http://xml.apache.org/xindice/>

¹⁶ <http://lucene.apache.org/>

die erhaltene Trefferliste erschöpfend ist, oder ob er noch gezielt den Suchstring variieren muss. In der online-Ausgabe werden ausschließlich exakte Übereinstimmungen gefunden.

Interessiert sich der Anwender für das Vorkommen eines Wortes ungeachtet der jeweiligen Flexionsstufe, so bietet sich eine lemmatisierte Suche an. Dies gelingt in FolioViews dann – aber auch nur dann - zufriedenstellend, wenn die flektierten Formen nicht zusätzlich noch orthographische Varianten enthalten. In der online-Ausgabe wird die lemmatisierte Suche derzeit noch nicht unterstützt. Eine weitere Möglichkeit ist bei dieser Fragestellung eine Suche mit Platzhaltern für einzelne Zeichen. FolioViews findet zuverlässig alle an dieser Stelle abweichenden Varianten, aber i. d. R. nicht alle Flexionen; der Platzhalter versagt außerdem bei manchen Sonderzeichen, etwa dem Bindestrich bei Komposita, die mal zusammen, mal mit Bindestrich geschrieben sind. Die online-Ausgabe findet zusätzlich auch Wörter, die sich in weiteren Stellen von der Eingabe unterscheiden und präsentiert dem Nutzer somit häufig eine unerwünscht umfangreiche Treffermenge. Die Ergebnisse bei Platzhaltern für beliebig viele Zeichen sind ähnlich.

FolioViews ist geeignet, um z. B. Wortfelder zu erschließen, etwa durch Verknüpfung mehrerer Suchbegriffe. Allerdings treten auch hier wieder Schwierigkeiten auf, sobald einer der gesuchten Begriffe Sonderzeichen enthält. Die verknüpften Begriffe müssen innerhalb eines Absatzes auftreten. In der online-Ausgabe wird die boolesche Verknüpfung der Suchbegriffe nicht unterstützt. Schließlich bietet FolioViews auch noch die Möglichkeit, den Bereich, innerhalb dessen die Suchbegriffe vorkommen müssen, auf beliebige Untereinheiten eines Absatzes einzuschränken.

Abgesehen von der wie beschrieben geringeren Mächtigkeit des Suchwerkzeuges in der online-Ausgabe fiel bei diesem störend auf, dass es maximal 100 Treffer anzeigt. Die Möglichkeit, sich alle Treffer auf einmal oder wenigstens in Paketen zu je 100 ausgeben zu lassen, wäre ebenso wünschenswert wie die Option, die Fundstellenliste wieder zu löschen.

3.6 TEI-Publisher

TEI-Publisher¹⁷ ist eine modulare Applikation für das Publizieren¹⁷ von TEI XML-Dokumenten im World Wide Web. Die Anwendung besteht – neben einem Installer – aus den zwei Kernkomponenten *teiWizard* und *teiRepository*. Der *teiWizard* bietet eine graphische Oberfläche zur Aufbereitung von XML-Dokumenten für die Darstellung im Web. Schrittweise kann der Benutzer Einstellungen für Suche und Anzeige der Dokumente vornehmen. Die so aufbereiteten Daten werden dann in einer Servlet-Umgebung, dem *teiRepository*, abgelegt.

3.6.1 Funktionalität

Der Publisher verwaltet TEI-Dokumente in einem Repository. XML-Dokumente und Dokument-Typ-Definitionen können in das Repository eingespeist oder daraus gelöscht werden. Durch ein Analysetool können für Suche und Anzeige im Web einzelne XML-Elemente aller im Repository liegenden Dokumente selektiert werden. Weitere Werkzeuge ermöglichen den Aufbau und die Verfeinerung einheitlicher Ontologien, Indizierung und Speicherung der XML-Dokumente für effiziente Suchen und Abfragen, interne und externe Verlinkung von Dokumenten und die Generierung der Anzeige für Suche, Suchergebnisse oder Metadaten für den Anwender. Um unterschiedliche Ausgabeformate (PDF, HTML) zu ermöglichen, lassen sich eigene XSLT- und CSS-Stylesheets einbinden.

3.6.2 Voraussetzungen

Die Installation des TEI-Publisher kann auf jeder beliebigen Plattform erfolgen, auf der ein Java Runtime Environment installiert ist. Die Anwendung basiert auf der XML-Datenbank *eXist* und der Bibliothek *Lucene*. Beide Systeme sollten entsprechend auf der Zielplattform installiert sein. Die Publikation kann auf jedem Java-Applikationsserver erfolgen, der Servlets unterstützt, beispielsweise *Tomcat*.

¹⁷ <http://teipublisher.sourceforge.net/docs/index.php>

3.6.3 Evaluierung

Der Komfort, den TEI-Publisher für Text-Retrieval mit sich bringt, spricht für einen Einsatz des Tools innerhalb von TextGrid. Beispielsweise ließen sich die Selektion der XML-Knoten für Anzeige und Suche, die Indizierung und das Deployment automatisieren. Die Möglichkeit, eigene XSLT- und CSS-Stylesheets zu hinterlegen, um damit unterschiedliche Ausgaben zu erzeugen, bietet die nötige Flexibilität für eine Integration.

Problematisch jedoch ist die Ausrichtung des Publishers auf die Verarbeitung von Dokumenten, die nach dem TEI P4-Standard kodiert sind, wodurch sich die Validierung der Dokumente auf Dokument-Typ-Definitionen beschränkt.

Für die weitere Nutzung wäre auf jeden Fall eine Anpassung an TEI P5 erforderlich. Der Aufwand dafür ist schwer abzuschätzen. Laut Sourceforge war die letzte Aktualisierung der Software im April 2005, die letzte Erwähnung im WWW Anfang 2006. Laut einer Auskunft von Susan Schreibman vom 23.03.2007 wird das Projekt vom Chefprogrammierer Amit Kumar nicht mehr weiterverfolgt. Der Quellcode steht aber nach wie vor zur Verfügung.

4 Zusammenfassung

Die Recherchemöglichkeiten in den TextGrid Datenbeständen werden mit darüber entscheiden, ob die im Projekt entwickelte Workbench von den Fachwissenschaftlern angenommen wird. Denn nur, wenn die teils komplexen fachwissenschaftlichen Fragestellungen mit einem vertretbaren (Lern-)Aufwand als konkrete Eingaben in das Recherchetool formuliert werden können und der Anwender darauf eine qualitativ hochwertige Antwort erhält, werden sich für die Wissenschaftler die Vorteile eines Datengrids erschließen. Dem stehen die erheblichen technologischen Herausforderungen gegenüber, die sich bei der Suche über verteilte, sehr große Datenbestände ergeben, speziell wenn auch noch Auszeichnungen, Kontext oder semantische Information berücksichtigt werden sollen.

Abhängig von den Vorkenntnissen des Anwenders und der Art der Anfrage (Volltextsuche, bezogen auf Auszeichnungen, Suche in Metadaten usw.) werden unterschiedliche Nutzerschnittstellen benötigt: Angefangen bei hierarchischen Browsern („Explorer“), über die TextGrid-Daten etwa anhand ihres Autors, ihres Entstehungsdatums oder ihrer Gattung und Titels zugänglich sind, über eine spartanische Volltextsuche vergleichbar Googles Suchmaske bis hin zu elaborierten Werkzeugen, die beliebige XQueries mit Metadaten und vielleicht auch noch semantischen Randbedingungen verknüpfen, werden diverse Nutzergruppen unterschiedliche Suchmasken benötigen. Auf die Implementierung von Ranking-Mechanismen wird vorerst verzichtet, zumal sie bei der linguistischen Suche das Ergebnis beeinträchtigen können.

Um der Heterogenität der mit TextGrid erstellten oder über TextGrid verfügbar gemachten Daten Rechnung zu tragen, kann TextGrid kein uniformes Auszeichnungsschema für Textdaten voraussetzen; die Anwender sind frei, ein Schema ihrer Wahl zu nutzen. Somit können Detailinformationen, die in diesem Schema verfügbar sind, für eine Recherche genutzt werden, wobei die Suche offensichtlich auf die so ausgezeichneten Texte eingeschränkt werden muss. Sollen auch übergreifende Recherchen möglich sein, so muss eine (möglicherweise verlustbehaftete) Abbildung des genutzten Schemas auf eines der von TextGrid definierten Kernschemata definiert sein. Werkübergreifende Recherchen können somit bezüglich eines Kernschemas formuliert und anhand einer Transformation der Werkauszeichnungen ins Kernschema bearbeitet werden.

Für die an TextGrid angebotenen Textarchive wird es verschiedene Integrationsstufen geben. In allen Fällen wird eine Suche in den Metadaten möglich sein; die Volltext- und Strukturdatensuche ist momentan aber nur für die in TextGrid selbst eingebrachten Daten realistisch. Für eine darüber hinaus gehende föderierte Suche muss noch die weitere technologische Entwicklung abgewartet werden.

In TextGrids Wörterbuch-Netzwerk sind Informationen über inhaltliche und zeitliche Begriffsbeziehungen hinterlegt, die semi-automatisch extrahiert werden können. Während eine weitergehende Erschließung der semantischen Relationen (z. B. im Sinne von Gegensätzen, Ober- und Unterbegriffen) wegen des dafür erforderlichen manuellen Aufwandes im Projektzeitraum nicht

machbar scheint, bieten sich die algorithmisch verfügbaren Beziehungen als wertvolle Unterstützung der Suche in TextGrid an, speziell bei der Recherche über mehrere Sprachstufen hinweg.

Für die TextGrid-Architektur ergibt sich aus den Anforderungen, dass sowohl eine Volltextindizierung der Dokumente als auch eine Strukturanalyse erforderlich ist, so dass verschiedene verfügbare Retrievaltools kombiniert werden müssen. Für die Volltextindexierung bietet sich Apache Lucene an; die Strukturdaten sollten, sofern sich in den Prototypen keine erheblichen Performanceprobleme zeigen, in einer XML-Datenbank – etwa eXist oder Berkeley DB-XML – hinterlegt werden. Zunächst ist jedoch zu prüfen, ob die von der eingesetzten XML-Datenbank generierten Indices für die Belange der TextGrid-Community ausreichend sind – zumal es mit einem gewissen Aufwand verbunden sein dürfte, Lucene XML-tauglich zu machen (s.o., 2.2.1.3 und 3.4).

Aus Zeitgründen und weil innerhalb der Community vorerst noch kein Bedarf angemeldet wurde, wird zunächst auf die Implementierung der in Abschnitt 2.2.5 (sprachspezifische Suche) besprochenen Techniken verzichtet.

Anhang A: Literatur

- [ATBC05] N. Aswani, V. Tablan, K. Bontcheva, H. Cunningham: *Indexing and Querying Linguistic Metadata and Document Content*. In: RANLP 2005, 21-23 September 2005, Borovets, Bulgaria.
- [BLE06] M. Baroni, A. Lüdeling, S. Evert,: *Using Web data for Linguistic Purposes*. In: Marianne Hundt, Caroline Biewer & Nadja Nesselhauf (eds): *Corpus Linguistics and the Web*. Rodopi, 7-24
- [Blo70] B. H. Bloom: *Space/Time Trade-offs in Hash Coding with Allowable Errors*. Commun. ACM Vol. 13(7), 422-426, 1970.
- [Bur95] L. Burnard: *Users' Reference Guide for the British National Corpus version 1.0*. Oxford: Oxford University Computing Services. 1995 <http://www.natcorp.ox.ac.uk/tools/index.xml>
- [CFK⁺06] S. Chernov, B. Fehling, C. Kohlschütter, W. Nejdl, D. Pieper, und F. Summan : *Enabling Federated Search with Heterogeneous Search Engines - Combining FAST Data Search and Lucene*. March 2006. <http://base.ub.uni-bielefeld.de/download/FedSearchReport.pdf>
- [Cil04] R. Cilibrasi, P. M. B. Vitanyi: *Automatic Meaning Discovery Using Google*. 2004. <http://www.arxiv.org/abs/cs.CL/0412098>
- [CSK94] O. Christ, B. M. Schulze, E. König,: *A Modular and Flexible Architecture for an Integrated Corpus Query System*. Proceedings of COMPLEX'94: 3rd Conference on Computational Lexicography and Text Research, Budapest, Hungary, 1994
- [CSK99] O. Christ, B. M. Schulze, E. König,: *Corpus Query Processor (CQP). User's Manual*. 1999. <http://www.ims.uni-stuttgart.de/CorpusWorkbench/CQPUserManual/HTML/>
- [DFA04] A. Dong, E. Fixler und A.M. Agogino,: *A Case Study of Policy Decisions for Federated Search Across Digital Libraries*. Proceedings of ICDL 2004 (International Conference on Digital Libraries), The Energy and Resources Institute, Vol. 2, 892- 898, 2004.
- [Dür98] M. J. Dürst (Hg.): *Requirements for String Identity Matching and String Indexing*, World Wide Web Consortium Working Draft 10-July-1998. <http://www.w3.org/TR/WD-charreq>
- [EJW98] K. Eibl, F. Jannidis, M. Willems: *Der junge Goethe in seiner Zeit*. Insel Verlag Frankfurt/Main 1998.
- [Küs01] M. W. Küster (Hg.): *Browsing and Matching (scoping)*. Project team report for CEN/TC304. Bruxelles: CEN, 2001. <http://std.dkuug.dk/CEN/tc304/matching/matchhome.html>

- [Lew05] D. Lewandowski: *Web Information Retrieval. Technologien zur Informationssuche im Internet*. Frankfurt am Main 2005, 248 Seiten mit Sachregister, DGI Schrift (Informationswissenschaft - 7), ISBN 3-925474-55-2.
- [MIRifs] Music Information Retrieval Gruppe, Technische Universität Wien: *Map of Mozart*. <http://www.ifs.tuwien.ac.at/mir/mozart/interaction.html>
- [Mit01] M. Mitzenmacher: *Compressed Bloom Filters*. In Proc. of the 20th ACM Symp. on Principles of Distributed Computing, pp. 144-150, 2001.
- [MOP02] J. Mirovský, R. Ondruška, D. Pruša: *Searching through Prague Dependency Treebank - Conception and Architecture*. In E. Hinrichs, K. Simov (Hg.): Proceedings of the Workshop on Treebanks and Linguistic Theories, (TLT02), Sozopol, Bulgaria, 2002. <http://www.bultreebank.org/proceedings/paper08.pdf>
- [RAK04] *Regeln für die alphabetische Katalogisierung in wissenschaftlichen Bibliotheken: RAK-WB*. Deutsches Bibliotheksinstitut. – Berlin. -Losebl.-Ausg. Stand 2004. Bezug: http://www.ddb.de/service/publikationen/publ_gedr.htm
- [SBC98] B. Shneiderman, D. Byrd, W.B. Croft: *Sorting Out Searching: A User-Interface Framework for Text Searches*. Communications of the ACM. ACM Press. Vol. 41. Issue 4. April 1998
- [Uni07] The Unicode Consortium: *The Unicode Standard, Version 5.0.0*. Defined by: *The Unicode Standard, Version 5.0*, Addison-Wesley, 2007.
- [Wil06] B.M. Wildemuth: *Evidence-based Practice in Search Interface Design*. [JASIST](#) 57(6): 825-828 (2006)