Technische Universität Kaiserslautern
Fachbereich Informatik
AG Datenbanken und Informationssysteme
Prof. Dr.-Ing. Dr. h.c. Theo Härder

Entwurf und Implementierung einer XML-Volltext-Suchmaschine

Diplomarbeit

von

Benedikt Eger

Betreuer:

Dipl.-Inf. Philipp Dopichaj

Mai 2005

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbständig und unter ausschließlicher Verwendung der angegebenen Literatur angefertigt habe.

Kaiserslautern, den 10. Mai 2005

Benedikt Eger

Inhaltsverzeichnis

Kapitel 1 Einführung

Die ständig wachsende Menge an elektronisch verfügbaren Informationen hat dazu geführt, dass immer größere Sammlungen von Dokumenten entstehen. Diese Informationen sind aber nur so lange wirklich nützlich wie es möglich ist, relevante Informationen daraus abzuleiten beziehungsweise relevante Dokumente darin zu finden. Um in der immer unüberschaubarer werdenden Masse nicht den Überblick zu verlieren, versucht man Methoden zu finden, wie man schnell und einfach die gewünschten Informationen extrahieren kann. Eine alphabetische Sortierung oder Schlagwortkataloge in Bibliotheken waren die ersten Lösungsansätze für dieses Problem.

Mit dem Aufkommen leistungsfähiger Rechner entstand die Idee, diese dazu zu benutzen, relevante Informationen leichter zu finden. Zu diesem Zweck gibt es schon seit den siebziger Jahren Versuche, den Vorgang der Suche nach relevanten Dokumenten zu automatisieren. Daraus entwickelte sich das Gebiet des Information Retrieval. Im Information Retrieval wird versucht, die Dokumente in eine geeignete Repräsentation zu überführen, die dann mit der Formulierung des konkreten Informationsbedarfs verglichen werden. Das Ziel dieser Bemühungen ist es, aus der Menge der Dokumente diejenigen herauszufinden, die den Informationsbedarf am Besten erfüllen, das heißt die relevant bezüglich des Informationsbedarfs sind. Die rasche Entwicklung des Internet hat das Interesse an Information-Retrieval-Systemen für die Suche im World Wide Web auch an die breite Öffentlichkeit gebracht.

In den letzten Jahren hat sich die Auszeichnungssprache XML auf breiter Front durchgesetzt, so dass immer mehr Dokumente entstehen, die mit XML strukturiert sind. XML ermöglicht es, beliebige Metadaten in die Dokumente zu integrieren. Durch den wachsenden Bestand derart ausgezeichneter Dokumente wuchs das Interesse, die Methoden des Information Retrieval auch auf diese Dokumente anzuwenden. Man erwartet, durch geeignete Interpretation der Metadaten eine Verbesserung des Retrieval zu erreichen.

Die Möglichkeiten, die durch XML-Dokumente geschaffen werden, führen jedoch dazu, dass die alten Konzepte nicht mehr in dieser Form anwendbar sind. Besonders die Möglichkeit, nun auch Teile der Dokumente als Ergebnis zu präsentieren, schafft viele neue und ungeklärte Fragen, die auch durch die aktuelle Forschung nur unzureichend beantwortet werden können.

Der jährliche Workshop der *Initiative for the Evaluation of XML Retrieval* (INEX) stellt seit 2002 ein Forum dar, bei dem neue Forschungsansätze präsentiert und verglichen werden. Die Ergebnisse dieses Workshops zeigen, dass sich noch kein Ansatz herauskristallisiert hat, der den anderen sichtbar überlegen ist. Daher besteht die Notwendigkeit, durch innovative Forschungsansätze neue Konzepte einzuführen, die die Struktur der XML-Dokumente gewinnbringend ausnutzen.

Ziel dieser Arbeit ist es, in das Gebiet des Information Retrieval auf XML-Dokumenen einzuführen und ein funktionsfähiges Information-Retrieval-System für XML-Dokumente zu erstellen. Dabei werden wir ein neues Konzept implementieren, das es gestattet, die in den Metadaten enthaltenen semantischen Informationen für eine Verbesserung des Retrieval zu nutzen.

In Kapitel 2 werden wir in die Grundlagen des traditionellen Information Retrieval einführen, die für das Verständnis der folgenden Kapitel die Basis darstellen. Dabei werden wir das dem Information Retrieval zugrundeliegenden Modell erklären, Repräsentationen für die Dokumente und Anfragen und verschiedene Modelle für die Suche beschreiben.

In Kapitel 3 unternehmen wir den Schritt zu XML-Dokumenten, indem wir zunächst eine Einführung in die Auszeichnungssprache XML geben. Dann beschreiben wir Konzepte und Lösungsansätze für das XML-Retrieval, bevor wir auf Möglichkeiten der Evaluierung solcher Systeme eingehen. Den Abschluss dieses Kapitels bildet eine Kategorisierung existierender Forschungsansätze und der Blick auf zwei etablierte XML-Retrieval-Systeme.

Den Hauptteil dieser Arbeit bildet der Entwurf und die Implementierung eines XML-Retrieval-Systems. Die dafür nötigen Konzepte, insbesondere das Kernkonzept, die Verwendung von Inline-Markup zur Verbesserung des Retrieval und eine geeignete Methode zum Zusammenfassen der Ergebnisse, erläutern wir in Kapitel 4. In Kapitel 5 werden wir die Details der Implementierung und die Verwendung der Benutzerschnittstellen beschreiben.

Die Arbeit schließt in Kapitel 6 mit einer Zusammenfassung, dem Versuch einer Bewertung und einem Ausblick auf weitere Forschung. Eine umfassende Bewertung unseres Systems auch im Vergleich mit anderen Systemen wird auf dem INEX-Workshop 2005 erfolgen.

Kapitel 2 Grundlagen des Information Retrieval

Mit der zunehmenden Menge an Informationen, die uns heute zur Verfügung steht, wird es immer wichtiger, effizienten Zugang dazu zu bekommen. Im Falle der elektronischen Speicherung von Informationen als Sammlungen von Dokumenten, im Folgenden als Dokumentenkollektion bezeichnet, muss ein Benutzer in der Lage sein, relevante Informationen abzufragen, ohne alle Dokumente einzeln einzusehen. Ein Information-Retrieval-System ist das Mittel, das den Benutzer dabei unterstützt, aus der großen Anzahl an Dokumenten diejenigen Informationen herauszusuchen, die für ihn interessant sind.

Dieses Kapitel führt in die Grundlagen des Information Retrieval (IR) ein. Zunächst werden einige Begriffe erklärt und das Gebiet des Information Retrieval in Aufbau und Modell definiert. Einem Abschnitt über die Möglichkeiten der Repräsentation von Dokumenten in einem Information-Retrieval-System folgt eine Einführung in die klassischen Retrievalmodelle. Das Kapitel schließt mit der Betrachtung einer der bekanntesten Methoden zur Evaluierung von IR-Systemen.

2.1 Begriffsklärungen

Die Fachgruppe „Information Retrieval“ der Gesellschaft für Informatik definiert ihren Aufgabenbereich, und damit das IR, folgendermaßen:

„Im Information Retrieval (IR) werden Informationssysteme in Bezug auf ihre Rolle im Prozess des Wissenstransfers vom menschlichen Wissensproduzenten zum Informations-Nachfragenden betrachtet. Die Fachgruppe ‚Information Retrieval‘ in der Gesellschaft für Informatik beschäftigt sich dabei schwerpunktmäßig mit jenen Fragestellungen, die im Zusammenhang mit vagen Anfragen und unsicherem Wissen entstehen. Vage Anfragen sind dadurch gekennzeichnet, dass die Antwort a priori nicht eindeutig definiert ist. Hierzu zählen neben Fragen mit unscharfen Kriterien insbesondere auch solche, die nur im Dialog iterativ durch Reformulierung (in Abhängigkeit von den bisherigen Systemantworten) beantwortet werden können; häufig müssen zudem mehrere Datenbasen zur Beantwortung einer einzelnen Anfrage durchsucht werden. Die Darstellungsform des in einem IR-System gespeicherten Wissens ist im Prinzip nicht beschränkt (z.B. Texte, multimediale Dokumente, Fakten, Regeln, semantische Netze). Die Unsicherheit (oder die Unvollständigkeit) dieses Wissens resultiert meist aus der begrenzten Repräsentation von dessen

Semantik (z.B. bei Texten oder multimedialen Dokumenten); darüber hinaus werden auch solche Anwendungen betrachtet, bei denen die gespeicherten Daten selbst unsicher oder unvollständig sind (wie z.B. bei vielen technisch-wissenschaftlichen Datensammlungen). Aus dieser Problematik ergibt sich die Notwendigkeit zur Bewertung der Qualität der Antworten eines Informationssystems, wobei in einem weiteren Sinne die Effektivität des Systems in Bezug auf die Unterstützung des Benutzers bei der Lösung seines Anwendungsproblems beurteilt werden sollte." [Fuhr96]

Es wird also versucht, aus gespeichertem Wissen in Form einer unstrukturierten Datenbasis den Informationsbedarf eines Benutzers zu befriedigen. Dabei spezifiziert der Benutzer seinen Informationsbedarf durch eine Anfrage, die die erwartete Antwort nur grob charakterisieren muss. Das IR-System versucht dann diejenigen Elemente der Datenbasis ausfindig zu machen, die relevant für diese Anfrage sind, wobei möglicherweise mehrere Benutzerinteraktionen notwendig sind, bis die endgültige Antwort vorliegt. Um sicherzustellen, dass die Antwort aus möglichst vielen relevanten und möglichst wenigen irrelevanten Elementen besteht, werden Bewertungsverfahren benötigt, mit denen sich Qualität der Antworten feststellen lässt.

Information Retrieval beschäftigt sich folglich mit der Repräsentation, Organisation und dem Zugriff auf Informationen [BYRN99]. Daher ist es für das Verständnis des IR wichtig zu klären, was unter dem Begriff „Information" eigentlich zu verstehen ist, insbesondere in Bezug auf die verwandten Begriffe „Daten" und „Wissen". Abbildung 1 zeigt die Beziehung zwischen den Begriffen.

Abbildung 1 Daten - Wissen - Information

Daten werden auf der syntaktischen Ebene, als reine Werte, von denen bestenfalls der Datentyp bekannt ist, angesiedelt. Sind diese Daten durch semantische Informationen ergänzt, dann spricht man von Wissen. Tabellen in einer Datenbank beispielsweise repräsentieren nach dieser Definition bereits Wissen, da durch den Namen der Tabelle und der einzelnen Spalten eine Interpretation der Daten in der Tabelle gegeben ist. Information ist *„die Teilmenge von Wissen, die von jemanden in einer konkreten Situation zur Lösung von Problemen benötigt wird"* [Kuhl90]. Wissen wird dadurch zu Information, dass aus dem Wissen für einen bestimmten Bedarf oder eine Anwendung ein Teil extrahiert wird, der diesen Bedarf erfüllt. Information repräsentiert also die pragmatische Ebene. Die Verwendung dieser Begriffe ist in der Literatur nicht ganz einheitlich, Fuhr [Fuhr04] zum Beispiel tauscht die Bedeutung von „Wissen" und „Information" aus.

Ein IR-System muss alle drei Aspekte integrieren, um am Ende Information gewinnen zu können.

2.2 Abgrenzung vom Fakten-Retrieval

Aus der Definition der Fachgruppe „Information Retrieval“ geht hervor, dass im IR der Begriff der Vagheit von zentraler Bedeutung ist. Dadurch lässt sich der Bereich des Information Retrieval vom Bereich des „Fakten-Retrieval“ (FR), zu dem zum Beispiel auch traditionelle Datenbanksysteme gehören, abgrenzen. Van Rijsbergen [Rijs79] führt eine Tabelle an, mit der er IR und FR in verschiedenen Eigenschaften voneinander unterscheidet (siehe Tabelle 1).

Tabelle 1 Abgrenzung des Information Retrieval vom Fakten-Retrieval nach van Rijsbergen

	Fakten-Retrieval	Information Retrieval
Matching	exakt	partiell, best match
Inferenz	Deduktion	Induktion
Modell	deterministisch	probabilistisch
Klassifikation	monothetisch	polythetisch
Anfragesprache	formal	natürlich
Fragespezifikation	vollständig	unvollständig
Gesuchte Objekte	die Fragestellung erfüllende	relevante
Reaktion auf Datenfehler	sensitiv	insensitiv

Es folgt eine kurze Erläuterung zu den einzelnen Punkten aus Tabelle 1. Zu den Begriffen, die später nochmal ausführlicher behandelt werden, finden sich Querverweise auf die entsprechenden Abschnitte:

- „*Matching*“: Im FR lässt sich für ein Element eindeutig bestimmen, ob es in der Ergebnismenge enthalten sein sollte oder nicht. Im IR kann auch eine teilweise Übereinstimmung mit der Anfrage ein zulässiges Ergebnis darstellen (Abschnitt 2.5).
- „*Inferenz*“: Deduktion bedeutet, dass beim FR nur Schlüsse gezogen werden, die notwendig wahr sind; Induktion erlaubt auch die Ableitung von Ergebnissen durch Verallgemeinerung.
- „*Modell*“: Die Einteilung in deterministisch beim FR und probabilistisch beim IR basiert auf der Wahrnehmung, dass durch die Vagheit zum Beispiel der Anfragen oder auch der Dokumente beim IR eine gewisse Unschärfe des Modells entsteht.
- „*Klassifikation*“: Eine monothetische Klassifikation kategorisiert Objekte danach, ob deren Attribute sowohl notwendig als auch hinreichend für die Zugehörigkeit zu einer Klasse sind. Im IR ist es aufgrund der unstrukturierten Datenbasis kaum möglich eine derart starre Einteilung der Elemente zu finden (Abschnitt 2.4.2).
- „*Anfragesprache*“: Im IR werden nach Möglichkeit die Anfragen als natürlichsprachliche Beschreibungen des Informationsbedarfs formuliert. Beim FR ist die Anfragesprache künstlich, mit beschränkter Syntax und Ausdrucksmöglichkeit. Bei der Suche in semistrukturierten

Dokumenten (siehe Abschnitt 3.2.2) wird meist eine Kombination dieser beiden Möglichkeiten verwendet.

- „*Fragespezifikation*“: Aus der Charakterisierung der Anfragesprache ergibt sich auch, dass im IR die Anfrage nicht erschöpfend die Elemente der Antwort beschreiben kann und, im Falle eines Relevanz-Feedback-Mechanismus, auch nicht unbedingt soll. Anfragen im FR beschreiben die erwarteten Ergebnisse vollständig.
- „*Gesuchte Objekte“:* Im FR werden Ergebnisse erwartet, die die Anfrage exakt erfüllen; beim IR möchte man die Objekte finden, die relevant für den Informationsbedarf des Benutzers sind (Abschnitt 2.6.1).
- „*Reaktion auf Datenfehler*“: Ein Fehler beim Abgleich ist beim FR kritisch, da dadurch möglicherweise ein die Fragestellung erfüllendes Element nicht in der Ergebnismenge enthalten ist. Die Ergebnisse im IR lassen sich ohnehin nicht so absolut abgrenzen wie beim FR, so dass kleinere Fehler hier tolerierbar sind.

Zusammenfassend lässt sich sagen, dass der wesentliche Unterschied darin besteht, dass beim FR bereits mit der Anfrage an das System exakt die Beschaffenheit der Ergebnisse definiert wird, so zum Beispiel bei einer SQL-Anfrage oder einem regulären Ausdruck. Diese exakte Verarbeitung wird dadurch ermöglicht, dass die Datenbasis für das FR einer streng definierten Form folgt. IR-Systeme hingegen versuchen die am Besten auf eine Anfrage passenden Informationen zu liefern. Ein Ergebnis, das nicht genau der Anfrage entspricht, ist im FR ein Totalausfall [BYRN99]; im IR, das ja auf unstrukturierten Daten arbeitet, ist so etwas nicht zu vermeiden und fällt möglicherweise auch nicht ins Gewicht.

Man sieht also, dass sich sowohl der Ansatz als auch das Anwendungsgebiet unterscheidet, obwohl sich beide Gebiete mit der automatisierten Gewinnung von Information beschäftigen. Die Unterscheidung ist aber nicht zwangsläufig als Gegensatz aufzufassen. Fuhr [Fuhr04] argumentiert, dass man die beiden Einträge in der Tabelle auch als Extrempunkte auf einer Skala auffassen kann, auf der es mehrere Abstufungen gibt, was zum Beispiel beim Retrieval auf semistrukturierten Daten der Fall ist.

2.3 Aufbau und formales Modell

Jedem IR-System liegt ein formales Modell zugrunde, mit dem die Anfragen der Benutzer mit den Dokumenten in der Kollektion verglichen werden.

Definition 1 **Information-Retrieval-Modell [BYRN99]**

Ein Information-Retrieval-Modell ist ein Viertupel [**D**, **Q**, $\mathcal{F}$, $R(q_i,d_j)$], wobei

D ist eine Menge bestehend aus logischen Sichten (bzw. Repräsentationen) der Dokumente in der Kollektion.

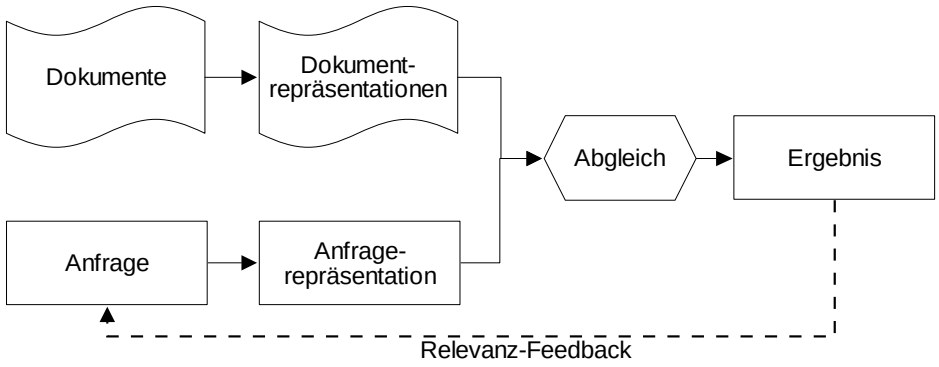
Q ist eine Menge bestehend aus logischen Sichten (bzw. Repräsentationen) für den Informationsbedarf der Benutzer, im Folgenden kurz als Anfragen bezeichnet.

$\mathcal{F}$ ist ein System für die Bestimmung der Dokumentrepräsentationen, der Anfragen und deren Beziehung zueinander.

$R(q_i,d_j)$ ist eine Bewertungsfunktion, die einer Anfrage $q_i \in$ **Q** und einer Dokumentrepräsentation $d_j \in$ **D** eine reelle Zahl zuweist. Diese Zahl bestimmt die Reihenfolge der Dokumente in Bezug auf die Anfrage q_i.

Aus dieser Definition eines IR-Modells kann man den allgemeinen Aufbau eines IR-Systems unabhängig von dessen verwendetem Modell ableiten (Abbildung 2).

Abbildung 2 **Aufbau eines IR-Systems**



Als Wissensbasis liegt zunächst eine *Dokumentenkollektion* vor. Ein Benutzer eines IR-Systems hat einen gewissen Informationsbedarf, von dem er annimmt, dass eine Teilmenge der Dokumente den Bedarf befriedigen kann. Der Benutzer muss nun versuchen, seinen Informationsbedarf möglichst präzise als *Anfrage* in der Anfragesprache des Systems auszudrücken.

Damit das IR-System automatisiert Entscheidungen über die Relevanz eines Dokumentes für eine Anfrage treffen kann, ist es notwendig sowohl für die Dokumentenkollektion als auch für die Anfragen, die an das System gestellt werden eine geeignete Abstraktion zu finden. Das bedeutet, die Dokumente und Anfragen müssen in eine *Dokumentrepräsentation* beziehungsweise *Anfragerepräsentation* überführt werden, bevor sie dem System für einen *Abgleich* zur Verfügung gestellt werden können. Als *Ergebnis* liefert das System dann idealerweise eine Bewertung zu den einzelnen Dokumenten oder alternativ die Dokumente sortiert nach dem Wert der Bewertungsfunktion.

In manchen IR-Systemen besteht zusätzlich für den Benutzer die Möglichkeit, die Ergebnisse einer ersten Suche zu bewerten, woraufhin das System die Anfrage modifiziert, um eine Verbesserung der Suchergebnisse zu erreichen. Eine solche interaktive Komponente nennt man *Relevanz-Feedback*.

2.4 Dokumentrepräsentationen

Um Dokumente automatisiert verarbeiten zu können, versucht man die Semantik der Dokumente in Repräsentationen umzusetzen, die die Inhalte vergleichbar machen. Dabei kann man zwei unterschiedliche Vorgehensweisen unterscheiden [Ferb03]:

1. Versuche, die natürliche Sprache so zu repräsentieren und zu verarbeiten, dass inhaltliche Ähnlichkeiten erkennbar werden;
2. Versuche, die zulässigen Mittel zur inhaltlichen Beschreibung so einzuschränken, dass sie Ähnlichkeiten abbilden.

In den folgenden Abschnitten wird zunächst der erste Ansatz, bei dem aus dem Dokument so genannte Indexterme isoliert werden, betrachtet. Anschließend wird der zweite Ansatz, die Klassifizierung von Dokumenten, behandelt.

2.4.1 Indexierung

Die Methode der Indexierung geht von der Prämisse aus, dass sich die Semantik von Dokumenten durch Indexterme repräsentieren lässt. Ein Indexterm ist im Wesentlichen ein Wort oder eine Wortgruppe, die eine semantische Bedeutung für das Dokument hat. Dies ist zwar eine starke Vereinfachung der Problemstellung, da dadurch die ursprüngliche Form des Dokumentes und somit auch ein Großteil der Semantik verloren geht, hat sich aber bei im Praxiseinsatz befindlichen IR-Systemen als sinnvoller Ansatz herausgestellt.

Die Indexierung geschieht in vier Schritten: Zunächst wird das Dokument in einzelne Zeichenketten zerlegt. Dann werden Wörter, die sehr häufig auftreten entfernt, die übrigen Wörter auf ihre Stammform reduziert, und diese Terme anschließend in einem Index gespeichert.

Syntaxanalyse

Das Dokument wird anhand vordefinierter Regeln in Zeichenketten zerlegt. Dabei können beispielsweise Satzzeichen, Ziffern oder sonstige Sonderzeichen aussortiert werden. Oft wird dabei auch der Text in Kleinschreibung überführt.

Stoppwortelimination

In jedem Dokument gibt es eine große Anzahl von Wörtern, die für sich allein genommen nichts zum Inhalt des Dokumentes beitragen. Daher empfiehlt es sich, diese Wörter nicht in den Index aufzunehmen, um ihn nicht unnötig zu vergrößern. Durch die Stoppwortelimination lässt sich der Umfang der Dokumente um 30 bis 50 Prozent verringern [Rijs79]. Es existiert normalerweise eine Stoppwortliste, die für eine bestimmte Sprache eine Liste solcher Wörter enthält. In Tabelle 2 findet sich je eine Liste für die deutsche und die englische Sprache.

Stammformreduktion

Mit der Stammformreduktion (*Stemming*) wird versucht, die Zeichenketten als bestimmte Formen desselben Wortes zu erkennen. Ziel davon ist es, unterschiedlich Formen desselben Wortstamms miteinander zu identifizieren, um zum Beispiel bei der Suche nach „*test*“ auch Dokumente zu finden, in denen „*testing*“, „*tests*“ oder „*tested*“ vorkommt. Die Wörter werden auf ihren linguistischen Stamm reduziert, der im Allgemeinen nicht mehr ein Wort der Sprache ist und für ein Verb und ein Substantiv identisch sein kann [Ferb03].

Ein weit verbreiteter Algorithmus ist der Porter-Stemming-Algorithmus [Port80], der ursprünglich für das Englische konzipiert wurde, von dem es aber mittlerweile Anpassungen für verschiedene andere Sprachen gibt. Dieser Algorithmus benutzt eine Menge aufeinander folgende Suffix-Ersetzungen bis eine Minimalzahl an Vokal-Konsonant-Sequenzen mit optionalen Konsonanten am Anfang und Vokalen am Ende erreicht ist. Beispiele für Suffix-Ersetzungen sind „*sses*“ nach „*s*“ wie in „*accesses*“, oder „*ies*“ nach „*i*“ und „*y*“ nach „*i*“, so dass „*libraries*“ und „*library*“ auf denselben Stamm „*librari*“ zurückgeführt werden [Henr99].

Für die deutsche Sprache ist die Stammformreduktion komplizierter. Probleme bereiten hier vor allem das Einfügen von Umlauten („*Baum*“ – „*Bäume*“), Präfixformen („*laufen*“ – „*gelaufen*“) und das häufige Auftreten von zusammengesetzten Wörtern. Hier reichen Suffixersetzungen nicht aus und man muss zu lexikonbasierten Morphologie-Analyse-Systemen greifen, die genauere Informationen zu den einzelnen Wortstämmen in einem Lexikon vorhalten [Ferb03].

Tabelle 2 Beispiele für Stoppwortlisten

Stoppwortliste Deutsch	Stoppwortliste Englisch
"einer", "eine", "eines", "einem", "einen", "der", "die", "das", "dass", "daß", "du", "er", "sie", "es", "was", "wer", "wie", "wir", "und", "oder", "ohne", "mit", "am", "im", "in", "aus", "auf", "ist", "sein", "war", "wird", "ihr", "ihre", "ihres", "als", "für", "von", "mit", "dich", "dir", "mich", "mir", "mein", "sein", "kein", "durch", "wegen", "wird"	"a", "an", "and", "are", "as", "at", "be", "but", "by", "for", "if", "in", "into", "is", "it", "no", "not", "of", "on", "or", "s", "such", "t", "that", "the", "their", "then", "there", "these", "they", "this", "to", "was", "will", "with"

Speicherung in invertierten Listen

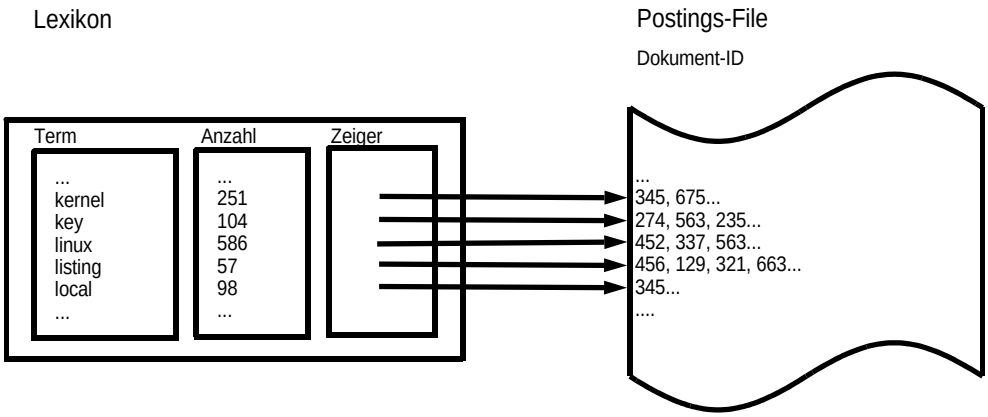
Die so vorbereiteten Indexterme müssen nun in einer geeigneten Datenstruktur abgelegt werden, die einen schnellen Zugriff auf die für die Suche nötigen Informationen erlaubt.

Von den verschiedenen Möglichkeiten für den Aufbau von Datenstrukturen für den Index wird hier nur die Speicherung in invertierten Listen behandelt, da dies für die meisten Anwendungen die bevorzugte Wahl sind.

Diese Speicherungsstruktur wird deshalb „invertiert“ genannt, da man eine Umkehr der normalen Speicherung von Wörtern mit dem Dokument vornimmt und zu jedem Wort beziehungsweise Indexterm eine Liste der Dokumente speichert, in denen der Term vorkommt.

Die invertierte Liste besteht aus zwei Teilen (siehe Abbildung 3). Im Lexikon werden die Terme zusammen mit der Gesamtanzahl des Auftretens dieses Terms in allen Dokumenten und einem Zeiger auf den ersten Eintrag für diesen Term im Postings-File abgelegt. Das Postings-File enthält Listen von Dokument-IDs, die jeweils für einen bestimmten Term alle Dokumente kennzeichnen, in denen dieser Term auftritt. Für bestimmte Suchoperationen beziehungsweise Suchmodelle werden manchmal noch zusätzliche Informationen wie die Vorkommenshäufigkeit im Dokument oder die Position des Vorkommens im Dokument benötigt [Henr99]. Zusätzlich gibt es noch ein Verzeichnis, das die Dokument-IDs wieder mit den Dokumenten assoziiert.

Abbildung 3 Aufbau einer invertierten Liste



Eine Speicherung der Terme in dieser Form gestattet einen sehr effizienten Zugriff auf die Terme und die Informationen über deren Auftreten in der Dokumentenkollektion.

2.4.2 Klassifikation

Klassifikationen spielen im Kontext dieser Arbeit eine eher geringe Rolle und werden hier nur zur Vollständigkeit erwähnt.

Mit einer Klassifikation lassen sich Dokumente systematisch ordnen. Man teilt dadurch die Menge der Dokumente in meist disjunkte Klassen ein, die in weiteren Schritten noch verfeinert werden können. Oft werden Dokumentklassifikationen manuell vorgenommen, wobei die einzelnen Klassen von vornherein feststehen müssen.

Speichert man den Bezeichner der Klasse eines Dokumentes als Attribut in einem IR-System ab, so kann bei einer Suche der Klassenname dazu dienen, die Ergebnisse auf einen bestimmten Themenbereich einzuschränken.

Beispiele für Klassifikationen sind thematische Einteilungen in Bibliotheken („Informatik“, „Physik“, „Mathematik“, „Wirtschaftswissenschaften“) oder auch die Systematik der Lebewesen in der Biologie („Ordnung“, „Familie“, „Gattung“, „Art“). Der große Nachteil von Klassifikationen ist, dass sie bedingt durch ihre Definition immer nur eine eingeschränkte Sichtweise auf die zu klassifizierenden Objekte zulassen.

2.4.3 Anfragen und Anfragerepräsentationen

Damit der Informationsbedarf des Benutzers für das IR-System handhabbar wird, muss der Benutzer diesen als Anfrage an das System formulieren. In IR-Systemen ist eine Anfrage entweder natürlichsprachig oder besteht aus eine Menge von Schlüsselwörtern, die den Informationsbedarf des Benutzers am treffendsten charakterisieren. Manche IR-Systeme unterstützen auch Anfragesprachen, mit denen sich komplexere Operationen ausführen lassen, wie beispielsweise einzelne Wörter von der Suche auszuschließen. Diese Anfrage wird dann durch äquivalentes Vorgehen wie bei den Dokumenten, also Syntaxanalyse, Stoppwortelimination und Stammformreduktion, in eine Anfragerepräsentation überführt.

Liegen Dokumente und Anfragen in ihrer jeweiligen Repräsentation dem IR-System vor, so kann mit dem Abgleich begonnen werden.

2.5 Konzepte für die Suche

Wie bereits erwähnt ist ein Suchergebnis im IR die Untermenge der Dokumentenkollektion, die den Informationsbedarf des Benutzers am Besten repräsentiert. Das heißt die Dokumente, die relevant bezüglich der Anfrage sind, sollen zurückgeliefert werden. Jegliche Suchstrategie basiert dabei auf dem Vergleich von Dokumentrepräsentationen und Anfragerepräsentation, führt also Operationen auf Mengen von Termen durch.

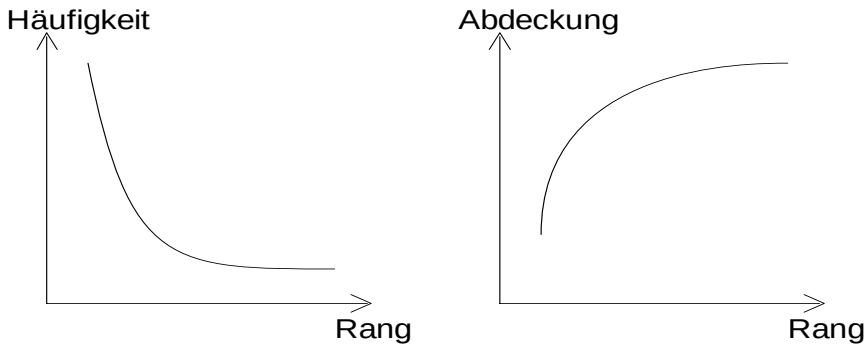
Allerdings sind nicht alle Terme bei der Suche von gleich großem Nutzen. Das Zipf’sche Gesetz beschreibt die Verteilung der Wörter innerhalb eines Textkorpus, also eines Dokumentes, einer Dokumentenkollektion oder einer Sprache (Abbildung 4 links):

Definition 2 **Zipf’sches Gesetz [Ferb03]**

Für einen repräsentativen Textkorpus C bezeichne W(C) die Menge der Wörter, die in C vorkommen, und h(w) die Häufigkeit, mit der das Wort $w \in W(C)$ in dem Korpus vorkommt. r(w) bezeichne den Rangplatz von $w \in W(C)$, wenn die Wörter nach abfallender Häufigkeit sortiert werden. Dann gilt

$$r \cdot h \sim c = \text{konstant } \forall (w \in W(C))$$

Abbildung 4 Bedeutung des Zipf'schen Gesetzen für die Wörter in einem Korpus



Das bedeutet, dass einige wenige sehr häufige Wörter einen großen Teil des Textes abdecken, wohingegen sehr viele, seltene Wörter nur einen geringen Teil ausmachen (Abbildung 4 rechts). Die Folge für IR-Modelle ist nun, dass manche Wörter einen größeren Beitrag zur Unterscheidung von Dokumenten leisten als andere. Ein Wort, das in allen Dokumenten vorkommt, taugt nicht zur Unterscheidung und wird daher sinnvollerweise in die Stoppwortliste aufgenommen (siehe Abschnitt 2.4.1). Ein Wort, das nur in bestimmten Dokumenten auftritt, eignet sich schon viel eher. Diese Erkenntnis sollte sich daher auch im Modell widerspiegeln, was man dadurch erreichen kann, dass man jedem Term k_i eines Dokumentes d_j eine Gewichtung $w_{i,j} > 0$ zuweist. Für einen Term, der nicht im Dokument d_j auftritt, ist $w_{i,j} = 0$. Man nimmt dabei vereinfachend an, dass die Gewichtungen paarweise unabhängig sind.

Diese Definition soll das Verständnis der IR-Modelle erleichtern, die im Folgenden besprochen werden. Die klassischen Modelle sind das boolesche Retrieval, das Vektorraumretrieval und das wahrscheinlichkeitstheoretische Retrieval, wobei die Beschreibung des Vektorraummodells wegen seiner Bedeutung für diese Arbeit etwas ausführlicher ist. Für diese Modelle gibt es mittlerweile zahlreiche Erweiterungen, die im Einzelnen hier aber nicht besprochen werden.

2.5.1 Boolesches Retrieval

Das boolesche Retrieval basiert auf der Mengenlehre und der booleschen Algebra. Eine Anfrage besteht aus Termen, die mittels der booleschen Operationen AND, OR und NOT verknüpft werden können. Die Suche liefert diejenigen Dokumente als Antwort zurück, für die die Auswertung des Ausdrucks den Wahrheitswert „wahr" zurückliefert. Der Wahrheitswert für einen Term definiert sich dadurch, ob dieser Term in einem Dokument enthalten ist oder nicht. Das bedeutet, für das boolesche Modell sind die Termgewichte binär, also $w_{i,j} \in \{0,1\}$.

Die Operationen lassen sich damit recht effizient auf die invertierten Listen abbilden. Die Suche zerlegt den unter Umständen sehr komplexen booleschen Ausdruck in das Problem der aufeinander folgenden Auswertung eines oder zweier miteinander verknüpfter Terme. Ein einzelner Term k_i wird auf die Liste l_i des Auftretens in den Dokumenten abgebildet, die ohne weitere Operationen aus dem Postings-File des Index gelesen werden kann. Bei einer AND-Verknüpfung zweier Terme k_i und k_j wird die Schnittmenge der Listen l_i und l_j verwendet, bei einer OR-Verknüpfung

die Vereinigungsmenge. Da eine NOT-Operation für einen Term k_i bedeuten würde, dass eine in vielen Fällen sehr umfangreiche Liste mit allen Dokumenten, in denen der Term nicht auftritt verwendet würde, geht man oft einen etwas anderen Weg. Erlaubt sind dann nur Anfragen der Art „k_j AND NOT k_i“, was sich effizienter implementieren lässt, indem man aus der Liste für das Vorkommen von k_j die Einträge der Liste für das Vorkommen von k_i entfernt.

Das boolesche Retrieval kann daher auch nur entscheiden, ob ein Dokument für eine Anfrage relevant erscheint oder nicht. Eine teilweise Übereinstimmung oder eine Rangordnung der Relevanz ist in diesem Modell nicht vorgesehen. In dieser Hinsicht ähnelt das boolesche Retrieval sehr stark dem Fakten-Retrieval. Weiterhin gibt es keine Möglichkeit, den Umfang der Antwortmenge einfach zu bestimmen. Trotz dieser großen Nachteile ist das boolesche Modell nach wie vor ein weit verbreitetes Verfahren, was sich durch die Einfachheit des Modells und die logische Nachvollziehbarkeit der Suchergebnisse begründet.

2.5.2 Vektorraumretrieval

Das Vektorraummodell unterstützt dagegen teilweise Übereinstimmungen bei der Suche und eine Bewertung der Relevanz der Ergebnisse. Man hat dadurch den Vorteil, dass die Ergebnisse nicht unsortiert zurückgegeben werden, was bei großen Ergebnismengen leicht sehr unübersichtlich wird. Stattdessen werden die den Informationsbedarf des Benutzers im Hinblick auf seine Anfrage am besten repräsentierenden Ergebnisse zuerst aufgeführt.

In diesem Fall werden sowohl den Termen aus den Dokumenten als auch denen aus der Anfrage positive und nicht-binäre Gewichte $w_{i,j}$ bzw $w_{i,q}$ zugeordnet. Ein Dokument d_j wird dann durch einen Vektor $\mathcal{D} = (w_{1,j}, w_{2,j},\ldots, w_{t,j})$ im t-dimensionalen Raum repräsentiert, wobei t die Gesamtanzahl der Terme im Lexikon des Systems ist. Ebenso wird auch die Anfrage q durch einen Vektor $\mathcal{Q} = (w_{1,q}, w_{2,q},\ldots, w_{t,q})$ repräsentiert. Die Ähnlichkeit zwischen einem Dokument d_j und einer Anfrage q wird durch eine Ähnlichkeitsfunktion, die auf den beiden Vektoren $\mathcal{D}$ und $\mathcal{Q}$ operiert, berechnet. Oftmals wird als Ähnlichkeitsfunktion der Cosinus des Winkels zwischen den beiden Vektoren angenommen. Der Cosinus eignet sich deshalb gut fur diesen Zweck, da er Werte zwischen 0 und 1 annimt, die direkt als Relevanz interpretiert werden können. Der Wert 0 resultiert aus der Berechnung des Cosinus bei keiner Übereinstimmung zwischen Anfrage und Dokument und der Wert 1 für vollständige Übereinstimmung. Bei teilweiser Übereinstimmung zwischen Anfrage und Dokument ergibt sich ein Wert, der je nach seiner Position im Intervall [0;1] den Grad der Übereinstimmung der beiden Vektoren angibt.

Damit berechnet sich die Ähnlichkeit wie folgt:

$$\mathrm{sim}(d_j, q) = \frac{\sum_{i=1}^{t} w_{i,j} \cdot w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \cdot \sqrt{\sum_{j=1}^{t} w_{i,q}^2}}$$

Nun geht es darum die Termgewichte $w_{i,j}$ derart zu berechnen, dass der damit gebildete Vektor $\mathcal{D}$ das Dokument möglichst gut beschreibt. D.h. es muss für jeden Term bestimmt werden, wie wichtig er für den Inhalt und die Unterscheidung eines Dokumentes ist. Man geht davon aus, dass zwei Faktoren die Relevanz eines Terms wesentlich beeinflussen:

Globale Gewichtungseinflüsse

Henrich [Henr99] bezeichnet globale Gewichtungseinflüsse als „*Trennschärfe des Begriffes bezogen auf die Dokumentenkollektion.*“ Man versucht eine Bewertung zu finden, wie gut ein Term ein Dokument von einem anderen in der Kollektion differenziert. Dabei ist ein Term um so besser zur Unterscheidung geeignet, wenn er nur in wenigen Dokumenten vorkommt. Man definiert die inverse Dokumenthäufigkeit *idf* (*inverse document frequency*) als Kehrwert der Anzahl der Dokumente, in denen dieser Term auftritt. Gewöhnlich wird die *idf* für einen Term k_i mit folgender Formel aus der Gesamtzahl N der Dokumente in der Kollektion und der Anzahl n_i der Dokumente, in denen der Term k_i vorkommt, berechnet: $idf_i = \ln\frac{N}{n_i}$

Lokale Gewichtungseinflüsse

Hier geht man von der Annahme aus, dass ein Term der häufig innerhalb eines Dokumentes auftritt eine größere semantische Bedeutung für den Inhalt des Dokumentes hat als ein seltener Term. Die Anzahl der Vorkommen innerhalb eines Dokumentes bezeichnet man als Termhäufigkeit *tf* (*term frequency*). Damit längere Dokumente, in denen die Terme im Allgemeinen eine höhere Häufigkeit haben, nicht bevorzugt werden, empfiehlt es sich zusätzlich, diesen Wert noch zu normieren. Eine Möglichkeit der Normierung ist, die Termhäufigkeit mit der Häufigkeit des häufigsten Terms in Beziehung zu setzen.

Globale und lokale Gewichtungseinflüsse werden zur Berechnung der Termgewichte in einer sogenannten *tf-idf-Formel* z.B. durch Multiplikation miteinander verknüpft.

Relevanz-Feedback

Darüberhinaus eignet sich das Vektorraummodell dazu, Relevanz-Feedback-Mechanismen zu implementieren. Durch Relevanz-Feedback lässt sich die Retrievalqualität weiter steigern. Angaben des Benutzers darüber, ob Dokumente aus der Ergebnismenge für ihn relevant sind oder nicht können dazu benutzt werden, die Gewichtung der Terme des Anfragevektors zu ändern. Man nimmt an, dass Dokumente, die eine ähnliche Thematik behandeln, im Vektorraum näher beieinander sind. Ein optimaler Anfragevektor, den man mit den vom Benutzer gelieferten Angaben über die Relevanz berechnet, zeigt dann vom Zentroiden der nicht relevanten Dokumente weg hin zum Zentroiden der relevanten Dokumente. Eine Erweiterung des Relevanz-Feedback, der Rocchio-Algorithmus, berücksichtigt auch den ursprünglichen Anfragevektor.

2.5.3 Wahrscheinlichkeitstheoretisches Retrieval

Der klassische Ansatz für das wahrscheinlichkeitstheoretische Retrieval wurde von Robertson und Sparck Jones im Jahr 1976 als *Binary-Independence-Retrieval-Modell* (BIR) [RoSJ76] formuliert.

Es basiert auf der Annahme, dass sich für eine Anfrage q die Menge aller Dokumente disjunkt in eine Menge R relevanter Dokumente und eine Menge NR der irrelevanten Dokumente einteilen lässt, es also eine sogenannte ideale Antwortmenge gibt.

Wenn man die Wahrscheinlichkeit $P(R \mid q,d_i)$ bestimmen kann, mit der ein Benutzer ein Dokument d_i für eine Anfrage q als relevant einstuft, stellen die Dokumente mit dem größten Wert die Antwortmenge dar.

Im BIR werden Anfragen q als Mengen von Index-Termen betrachtet. Ein Dokument wird durch einen binären Vektor $\mathcal{D} = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$ mit $w_{i,j} \in \{0,1\}$ repräsentiert. Das Ähnlichkeitsmaß wird daraus wie folgt berechnet:

$$sim(q, d_i) = \frac{P(R \mid q, D)}{P(NR \mid q, D)}$$

Das eigentliche Modell wird aus dieser Formel unter anderem unter Verwendung des Bayes'schen Theorems abgeleitet. Einzelheiten dazu finden sich in [Fuhr04].

Das Problem ist nun die Bestimmung dieser Wahrscheinlichkeiten. Zu Beginn wird eine Wahrscheinlichkeit für die Werte durch einige grobe Annahmen abgeschätzt. Aus der Bewertung des Benutzers für diese initiale Antwortmenge ermittelt das System dann neue Werte für die Wahrscheinlichkeiten, die die Menge der relevanten Dokumente genauer beschreiben. Dieser Vorgang wird iterativ fortgesetzt, bis das System die ideale Antwortmenge genau beschreibt.

Leider hat dieses Modell den Nachteil, dass gerade die erste Einteilung der Dokumentenkollektion in relevante und irrelevante Dokumente geschätzt werden muss. Auch die Verwendung von binären Termgewichten $w_{i,j}$ ist eine eher unrealistische Annahme.

2.6 Bewertungsverfahren

Damit man die Qualität eines IR-System messen oder verschiedene IR-Systeme miteinander vergleichen kann, benötigt man gewisse Bewertungsmaßstäbe. Dabei wird meist zwischen den zwei Gebieten *Effizienz* und *Effektivität* unterschieden.

Effizienz bezeichnet die in der Informatik üblichen Metriken in Bezug auf Systemressourcen, wie zum Beispiel Speicherplatz, Prozessorbelastung oder Antwortzeiten. Obwohl natürlich die Effizienz eines Systems eine große Rolle für den praktischen Einsatz darstellt, wird auf eine genauere Betrachtung dieses Gebietes hier verzichtet. Zur Bewertung der Effizienz können Standardmethoden der Softwareentwicklung verwendet werden.

Effektivität ist ein Maß für die Fähigkeit des Systems, den Benutzer zufrieden zu stellen. Bewertet wird die Qualität des Ergebnisses im Verhältnis zum Aufwand, der bei der Benutzung des Systems entsteht. Van Rijsbergen [Rijs79] definiert Effektivität als *„die Fähigkeit des Systems, die relevanten Dokumente abzurufen, während gleichzeitig die irrelevanten zurückgehalten werden.*"

Im Folgenden wird genauer auf die Maße *Recall* und *Precision* eingegangen, die das am häufigsten verwendete Kriterium zur Bewertung der Effektivität eines IR-Systems sind. Zuvor soll jedoch der Begriff der Relevanz etwas genauer beleuchtet werden.

2.6.1 Relevanz

Ziel jedes IR-Systems ist es, für einen gewissen Informationsbedarf diejenigen Dokumente abzurufen, die relevant für diesen Informationsbedarf sind. Leider ist Relevanz ein sehr subjektiver Begriff, der stark vom jeweiligen Benutzer abhängig ist. Vereinfachend wird daher angenommen, dass es für eine bestimmte Anfrage möglich ist, die Menge der Dokumente präzise in relevante und irrelevante Dokumente zu teilen.

Um ein IR-System bewerten zu können, müssen für die von ihm gelieferten Ergebnisse Informationen über die Relevanz vorliegen. Da das in der Praxis wohl kaum der Fall sein dürfte, behilft man sich mit vordefinierten Sammlungen von Dokumenten und Anfragen.

Zu einer festgelegten Dokumentenkollektion werden Anfragen formuliert, für die dann von Experten auf diesem Gebiet manuell Relevanzbewertungen zu den Dokumenten vorgenommen werden. Die Anfragen werden dann auch an das System gestellt und die Relevanzbewertung des Systems mit den manuell erstellten verglichen. Man nimmt an, dass ein IR-System, das sich unter diesen experimentellen Bedingungen bewährt, auch im Praxiseinsatz effektiv ist.

2.6.2 Recall und Precision

Für die Bewertung mittels Recall und Precision teilt man die Menge der Dokumente für eine bestimmte Anfrage in folgende Teilmengen [Henr99]:

- relevant und im Ergebnis enthalten (*hits*)
- nicht relevant und trotzdem im Ergebnis (*noise*)
- relevant und trotzdem nicht im Ergebnis (*misses*)
- nicht relevant und auch nicht im Ergebnis (*rejected*)

Recall ist dann definiert als

$$\frac{\text{hits}}{\text{hits} + \text{misses}}$$

also der Anzahl der relevanten Dokumente im Ergebnis im Verhältnis zur Gesamtzahl der relevanten Dokumente.

Precision ist definiert als

$$\frac{\text{hits}}{\text{hits} + \text{noise}}$$

also die Anzahl der relevanten Dokumente im Ergebnis im Verhältnis zur Gesamtzahl der Dokumente im Ergebnis.

Recall kann nach Henrich [Henr99] auch betrachtet werden als Antwort auf die Frage „*Wie vollständig ist die Antwort?*“; Precision als Antwort auf „*Wie genau ist die Antwort?*“.

Diese beiden Zahlen lassen sich für das boolesche Retrieval relativ genau bestimmen, da dort nur über Relevanz und Irrelevanz entschieden wird. Sobald es aber eine Bewertung der Ergebnisse gibt, ergeben sich unterschiedliche Werte für Recall und Precision, je nachdem wie weit man die Liste der Ergebnisse betrachtet.

Beispiel 1 **Schrittweise Berechnung von Recall und Precision**

Folgendes Beispiel soll diesen Zusammenhang verdeutlichen. Wir nehmen eine Anfrage an, zu der es in einer bestimmten Dokumentenkollektion 10 relevante Dokumente gibt. Das System liefere eine Ergebnisliste mit 10 Ergebnissen, die folgendermaßen verteilt sind, wobei „R“ für ein relevantes Dokument, „U“ für ein nicht relevantes Dokument steht:

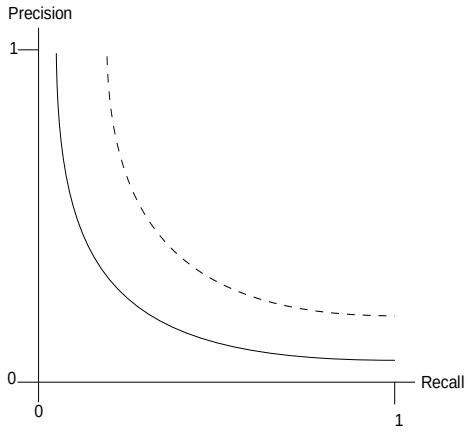
R U R R U R R R U U

Dann zeigt die Tabelle die Werte für Recall und Precision, je nachdem wie weit man diese Liste betrachtet.

Berechnung bis Position	1	2	3	4	5	6	7	8	9	10
Recall	0,1	0,1	0,2	0,3	0,3	0,4	0,5	0,6	0,6	0,6
Precision	1	0,5	0,66	0,75	0,6	0,66	0,71	0,75	0,66	0,6

Wie in Abbildung 5 ergibt sich eine Evaluierung damit nicht als einzelner Wert, sondern als Kurve beziehungsweise Menge von Punkten. Normalerweise berechnet man die Werte für mehrere Anfragen und mittelt dann das Ergebnis, um eine aussagekräftige Evaluierung des IR-Systems zu erhalten.

Abbildung 5 **Precision-Recall-Diagramm**



Die Bewertung mit Recall und Precision ermöglicht auch den Vergleich verschiedener IR-Systeme untereinander. Man bestimmt zu diesem Zweck meist an vorgegebenen Recall-Punkten den Precision-Wert für die unterschiedlichen Systeme. Eine gebräuchliche Methode ist die sogenannte “*Eleven-Point Average Precision*”, bei der ein Durchschnittswert für die Precision bei den Recall-Werten 0,0, 0,1, 0,2 bis 1,0 gebildet wird. Aussagen darüber, ob ein Verfahren besser ist als ein anderes können nur dann zuverlässig getroffen werden, wenn für ein Recall-Precision-Paar beide Werte besser sind, was für Precision-Recall-Diagramme bedeutet, dass eine Kurve

vollständig oberhalb der anderen liegen muss. Demnach liefert in Abbildung 5 das System mit der gestrichelten Kurve qualitativ bessere Ergebnisse als das mit der durchgezogenen Kurve.

Kapitel 3 XML Information Retrieval

Das traditionelle Information Retrieval, das im letzten Kapitel vorgestellt wurde, beschäftigt sich nur mit Dokumentenkollektionen, die flache, also unstrukturierte Dokumente enthalten. Dabei wird in den Dokumenten keine weitere Einteilung in Kapitel oder Abschnitte vorgenommen, das heißt die Dokumente werden als Block betrachtet. In diesem Kaptiel gehen wir nun einen Schritt weiter und betrachten das Information Retrieval auf semistrukturierten Dokumenten.

Semistrukturierte Dokumente haben auch und vor allem durch die Popularität des Internet in den letzten Jahren an Bedeutung zugenommen. Speziell der Standard XML (eXtensible Markup Language) [YBP+04] hat sich als Auszeichnungssprache durchgesetzt. Eine Auszeichnungssprache dient dazu, innerhalb eines Dokumentes Informationen zu speichern, die über den reinen Inhalt hinausgehen.

Dadurch wird es möglich Dokumente in mehrerlei Hinsicht zu strukturieren. Einerseits lässt sich die *Syntax* der Dokumente auszeichnen, zum Beispiel einzelne Absätze, Kapitelüberschriften oder Kapitel lassen sich als solche im Dokument markieren, aber auch semantische Informationen, zum Beispiel darüber, ob eine Textpassage ein Dateipfad oder eine Abkürzung ist, können in das Dokument aufgenommen werden. Andererseits kann man *Metadaten* in den Dokumenten unterbringen, die beispielsweise Informationen wie den Namen des Autors oder das Datum der Veröffentlichung des Dokumentes enthalten.

Durch die große Verbreitung von XML und die Überzeugung, dass sich XML als ein universelles Format durchsetzen wird, entstand schnell der Wunsch, große Kollektionen von XML-Dokumenten nicht nur effektiv und effizient ordnen und abrufen zu können. Man wollte außerdem in der Lage sein, die Struktur der XML-Daten dazu auszunutzen, um die Suche präziser zu machen.

Im Juli 2000 fand in Athen die ACM-SIGIR-Konferenz statt, wo Carmel, Maarek und Soffer einen ersten Workshop zum Thema „IR und XML“ veranstalteten. Die wichtigsten Themen dieses Workshops waren die Fragen,

- wie man bestehende IR-Technologien zur Suche in XML-Dokumenten erweitern kann,
- wie die Struktur der XML-Dokumente indexiert werden kann, damit man sowohl im Inhalt als auch in der Struktur der Dokumente suchen kann
- und wie semantische Informationen innerhalb der Dokumente zum Verbessern der Suche eingesetzt werden können [CaMS00].

Seit 2002 findet jährlich der INEX-Workshop statt, der sich ausschließlich mit der Präsentation und Evaluierung aktueller Forschungsergebnisse auf dem Gebiet des XML-Retrieval beschäftigt.

Dieses Kapitel beginnt mit einer Einführung in die Grundlagen von XML und darauf aufbauenden Anfragesprachen XPath und XQuery sowie dem Metadatenmodell RDF, die im Zusammenhang mit Information Retrieval auf XML-Dokumenten wichtig sind. Der nächste Abschnitt beschäftigt sich dann mit den verschiedenen Möglichkeiten, die die Struktur der XML-Dokumente zur Indexierung und Suche bietet und den damit verbundenen Problemstellungen. Der darauf folgende Abschnitt widmet sich den Überlegungen, die bei der Bestimmung der Ergebnisse eine Rolle spielen. Danach wird anhand der bei INEX verwendeten Verfahren auf die Unterschiede in der Evaluation der Ergebnisse eingegangen, die sich bei XML-Dokumenten im Vergleich mit den traditionellen Methoden ergeben, bevor dann abschließend ein Überblick über die verschiedenen Forschungsansätze gegeben wird und zwei konkrete Projekte, HyREX und XXL, vorgestellt werden.

3.1 XML Grundlagen

XML wird vom World Wide Web Consortium (W3C) in [YBP+04] definiert. Mit XML lassen sich strukturierte Dokumente erstellen, die sowohl von Menschen als auch automatisiert interpretierbar sind. XML wurde aus dem ISO-Standard SGML (Standard Generalized Markup Language) abgeleitet, so dass jedes XML-Dokument auch ein standardkonformes SGML-Dokument ist.

Wichtige Aspekte von XML sind die Möglichkeit, beliebige Elemente selbst zu definieren und die damit verbundene Eigenschaft, selbstbeschreibend zu sein, d.h. sowohl Daten als auch Metadaten in einem Dokument zu vereinen. Dieser Abschnitt führt in die wichtigsten Grundlagen von XML ein und betrachtet auch die für das IR wichtigen Standards XPath und XQuery sowie XPath und XQuery Full-Text. Abschliessend gehen wir kurz auf das Metadatenformat Resource Description Framework (RDF) ein.

3.1.1 XML und DTD

XML wurde 1998 als Metasprache vom W3C entwickelt, um Datenbeschreibungssprachen für den elektronischen Dokumentenaustausch definieren zu können. Ein XML-Dokument wie in Beispiel 2 besteht aus Textauszeichnung (*Markup)* und Text. Im Einzelnen sind dies folgende Komponenten:

XML-Dokumente beginnen mit einer XML-Deklaration, die als erste Zeile im Dokument auftaucht und die verwendete XML-Version angibt, im Beispiel `<?xml version=”1.0”?>`.

Jedes XML-Dokument enthält ein oder mehrere *Elemente*, die beliebig tief geschachtelt sein dürfen. Jedes Element hat einen Typ, der durch dessen Namen festgelegt wird. Das Element beginnt mit einem *Start-Tag*, im Wesentlichen der Name in spitzen Klammern, und endet mit einem *End-Tag*, der Name des Elementes in spitzen Klammern mit vorangestelltem Schrägstrich. In Beispiel 2 ist also „*author*“ ein Element, `<author>` dessen Start-Tag und `</author>` der End-Tag. Leere Elemente können außerdem durch einen Schrägstrich vor der schließenden spitzen Klammer gekennzeichnet (z.B. `<empty />`) werden. Der End-Tag entfällt in diesem Fall. Alle Elemente müssen korrekt geschachtelt sein; die Verschachtelung von Elementen ineinander in der Form `<a><b></a></b>` ist nicht gestattet. Außerdem muss ein Wurzelelement existieren,

das alle anderen Elemente umschließt. In Beispiel 2 ist „*book*“ das Wurzelelement. Wenn im Folgenden die Rede von *XML-Fragmenten* ist, so bezeichnet dies ein Element und dessen kompletten Inhalt.

Alle Elemente können *Attribute* in Form von Name-Wert-Paaren enthalten. Attribute werden dabei innerhalb des Start-Tags angegeben. In Beispiel 2 hat das Element „*book*“ ein Attribut `firstPublished` mit dem Wert `1979`.

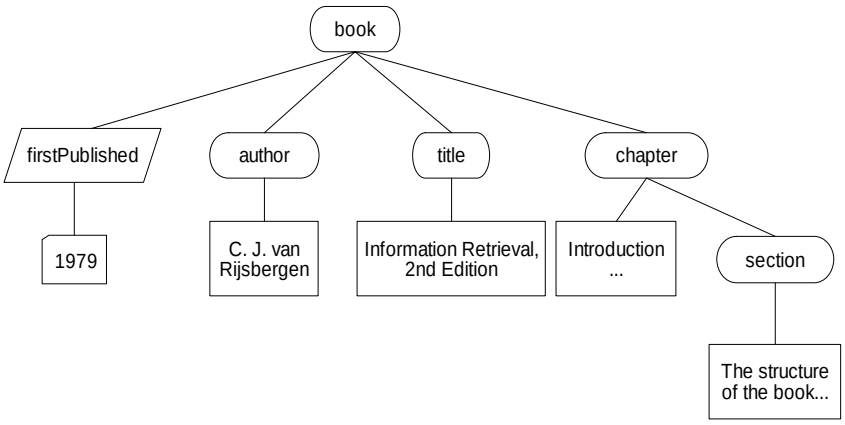
Beispiel 2 **XML Beispieldokument**

```
<?xml version=”1.0”?>
<!DOCTYPE book SYSTEM "book.dtd">
<book firstPublished=”1979” >
    <author>C. J. van Rijsbergen</author>
    <title>Information Retrieval, 2nd Edition</title>
    <chapter>Introduction...
        <section>The structure of the book...</section>
        ...
    </chapter>
</book>
```

Die eigentlichen Daten sind innerhalb eines XML-Dokumentes in Text-Abschnitten enthalten, wobei innerhalb eines Elementes Text und andere Elemente als „*mixed content*“ auch gemeinsam auftreten können (z.B. „`<p>Text gemeinsam mit <em>Elementen</em></p>`“).

Der Aufbau in dieser Form ausgezeichneter Dokumente läßt sich als eine Baumstruktur begreifen, wobei jeder Knoten ein Element, ein Attribut oder dessen Wert, oder Text sein kann (Abbildung 6). Elementknoten haben als Kindknoten alle Elemente und den Text, die sie umschließen, sowie ihre Attribute.

Abbildung 6 **Baumstruktur des XML-Dokuments aus Beispiel 2**



Zusätzlich kann ein XML-Dokument eine Dokumenttyp-Deklaration enthalten, die auf eine Document Type Definition (DTD) verweist. Zeile 2 aus Beispiel 2 verweist auf eine DTD „*book.dtd*“, die in Beispiel 3 abgedruckt ist. Die DTD definiert die Grammatik des XML-Dokumentes, also zum Beispiel welche Elemente enthalten sind und wie die Elemente geschachtelt werden dürfen. Außerdem lässt sich vorschreiben, welche Attribute die Elemente haben. Beispiel 3 definiert, dass innerhalb des „*book*“-Element mindestens drei Elemente, ein „*author*“- und ein „*title*“-Element und mindestens ein „*chapter*“-Element, verwendet werden dürfen. Innerhalb dieser Elemente ist nur Text („*#PCDATA*“, *parsed character data*) erlaubt, mit Ausnahme des „*chapter*“-Elements, das Text und beliebig viele „*section*“-Elemente enthalten darf. Das „*section*“-Element darf wiederum nur Text enthalten. Außerdem wird definiert, dass das Element „*book*“ ein Attribut mit dem Namen „*firstPublished*“ hat.

Beispiel 3 **DTD zum Dokument aus Beispiel 2**

```
<!ELEMENT book (author, title, chapter+)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT chapter (#PCDATA|section*)>
<!ELEMENT section (#PCDATA)>

<!ATTLIST book firstPublished CDATA>
```

Eine andere Möglichkeit der Strukturdefinition bietet XML Schema [FaWa04]. XML Schema gestattet eine komplexere Definition der erlaubten Dokumentstruktur als es durch DTD möglich ist, beispielsweise können Datentypen definiert und als Beschränkung für Attribute oder Elementinhalte verwendet werden. XML Schema ist jedoch weit weniger gebräuchlich.

Das W3C definiert außerdem die Begriffe „*wohlgeformtes*“ („*well-formed*“) und „*gültiges*“ („*valid*“) XML für das Arbeiten mit XML-Dokumenten. Wohlgeformt ist ein XML-Dokument, wenn es sich an die in diesem Abschnitt besprochenen Regeln für die Syntax hält. Wenn in einem wohlgeformten XML-Dokument eine DTD angegeben ist, und das Dokument der in der verwendeten DTD angegebenen Grammatik entspricht, so ist das XML-Dokument auch gültig.

Weiterhin unterscheidet man zwischen einer daten- und dokumentorientierten Sicht. Die datenorientierte Sicht betrachtet XML lediglich als Austauschformat für strukturierte Daten, wohingegen die dokumentorientierte Sicht XML als Format zur Repräsentation der logischen Struktur von Dokumenten interpretiert. In diesem Fall spricht man auch von semistrukturierten Dokumenten. Ausser an Stellen, wo explizit darauf hingewiesen wird, gehen wir in dieser Arbeit von einer dokumentorientierten Sicht aus.

3.1.2 XML-Anfragesprachen

Um Informationen aus XML-Dokumenten abfragen zu können, hat das W3C die Anfragesprachen XPath und darauf aufbauend XQuery entwickelt. Beide Sprachen beschränken sich darauf, im Sinne des Fakten-Retrieval (siehe Abschnitt 2.2) Informationen aus XML-Dokumenten abzufragen. Mit der Spracherweiterung XQuery beziehungsweise XPath Full-Text wird versucht, diese Beschränkung durch die Einführung von Information-Retrieval-Elementen zu überwinden.

XPath

XPath [ClDe99] ist eine Sprache, mit der Teile von XML-Dokumenten adressiert werden können. XPath betrachtet die Struktur der Dokumente als Baumstruktur, daher ist auch die Syntax der von Verzeichnispfaden im Dateisystem ähnlich. XPath unterscheidet verschiedene Typen von Knoten, zum Beispiel Element-, Attribut- und Textknoten. Betrachtet man wieder das XML-Dokument aus Beispiel 2, so lässt sich mit dem XPath-Ausdruck „`/book`“ der Wurzelknoten auswählen. „`/book/author`“ selektiert alle „*author*“-Elemente unterhalb des „*book*“-Elements. In unserem Beispiel existiert nur ein einziger, so dass dieser Ausdruck dasselbe bewirkt wie der Ausdruck „`/book/author[1]`“, der nur das erste „*author*“-Element selektiert. Um Attribute zu adressieren verwendet man einen Ausdruck ähnlich diesem, der das „*firstPublished*“-Attribut anspricht: „`/book@firstPublished`“. XPath erlaubt auch die Verwendung von Wildcards („`/book/*`“) oder Restriktionen auf einzelnen Attributen oder Elementen („`/book[@firstPublished=”1979”]`“). Weiterhin sind relative Pfade vom aktuellen Kontextelement erlaubt („`./author`“) und die globale Selektion bestimmter Elemente unabhängig von deren Pfad („`//title`“).

XQuery

XQuery [BCF+04] ist eine Anfragesprache zum Abfragen großer XML-Dokumentenkollektionen. Sie verwendet XPath zur Adressierung innerhalb von Dokumenten und die sogenannte FLWOR-Syntax für die Anfragen. FLWOR ist eine Abkürzung für “*for*, *let*, *where*, *order by*, *return*”. Die *for*- und *let*-Klausel in den Anfragen erzeugen eine geordnete Sequenz von Tupeln gebundener Variablen, die *Tuple-Stream* genannt wird. Mit der *where*-Klausel kann man den Tuple-Stream filtern, mit der *order-by*-Klausel umordnen. Die *return*-Klausel erzeugt die Ausgabe der Ergebnisse.

XQuery und XPath Full-Text

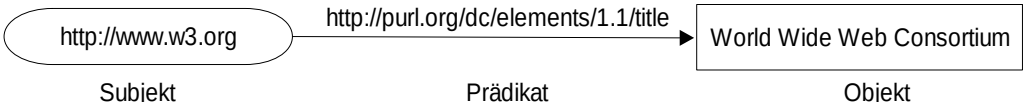
Im Juli 2004 veröffentlichte das W3C den ersten öffentlichen Entwurf von XQuery und XPath Full-Text [AYB+04], einer Spracherweiterung für XQuery und XPath, die Elemente des Information Retrieval in die beiden Anfragesprachen integriert. Die wichtigsten Neuerungen beinhalten boolesche Anfragen zur Wort- und Phrasensuche, Möglichkeiten zur Stoppworteliminierung und Stammformreduktion. Außerdem sollen die Spracherweiterungen eine Bewertung der Suchergebnisse nach Relevanz ermöglichen. Mit dem GalaTex-System [CABF04] existiert bereits die erste Referenzimplementierung von XQuery und XPath Full-Text.

3.1.3 Das Metadaten-Modell RDF

RDF (*Resource Description Framework*) [MaMi04] ist eine Sprache zur Repräsentation jeglicher Art von Metadaten. RDF basiert auf der Idee, Dinge durch URIs (*Uniform Resource Identifiers*) zu identifizieren und Resourcen durch Subjekt-Prädikat-Objekt-Ausdrücke zu beschreiben. Das Subjekt ist die zu beschreibende Resource, das Prädikat die Eigenschaft des Subjekts und das Objekt der entsprechende Wert. Man unterscheidet bei der Darstellung zwischen dem RDF-Modell, das durch einen Graph repräsentiert wird und der RDF-Syntax, einer Serialisierung des Graphen in XML. Beispiel 4 zeigt die XML-Darstellung der Eigenschaft „*title*“ der Resource „*http://www.w3.org*“ mit dem Wert „*World Wide Web Consortium*“ und den zugehörigen Graph.

Beispiel 4 **RDF-XML-Syntax und Darstellung als Graph**

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about="http://www.w3.org/">
        <dc:title>World Wide Web Consortium</dc:title>
    </rdf:Description>
</rdf:RDF>
```



3.2 Indexierung und Suche auf XML-Dokumenten

Die Möglichkeiten, die XML zur Auszeichnung von Dokumenten bietet, haben die Hoffnung geweckt, dass man die strukturellen, selbstbeschreibenden Informationen und die Metadaten innerhalb der Dokumente verwenden kann, um die Ergebnisse bei der Suche innerhalb von XML-Dokumentenkollektionen zu verbessern.

Im folgenden werden zunächst Methoden für die Indexierung von XML-Dokumenten nach Luk et al. [LLD+02] beschrieben. Dann folgt ein Abschnitt über Implikationen für die Suche.

3.2.1 Indexierung

Methoden für die Indexierung von XML-Dokumenten lassen sich grob in die Kategorien „*flache Dateien*“, „*semistrukturiert*“ und „*strukturiert*“ einteilen. Wichtig ist hierbei die Unterscheidung zwischen strukturierten beziehungsweise semistrukturierten Dokumenten und strukturierter beziehungsweise semistrukturierter Indexierung. Die Begriffe strukturierte und semistrukturierte Indexierung beziehen sich darauf, in welchem Maß die in den Dokumenten enthaltene Struktur, sei diese selber nun strukturiert oder semistrukturiert, indexiert wird.

Flache Dateien

Die einfachste Möglichkeit zur Indexierung von XML-Dokumenten ist, die Dokumente als flache Dateien zu betrachten (*flat-file indexing*) und sie dann mit den traditionellen IR-Methoden wie in Kapitel 2 zu verarbeiten.

Eine Variante ist dabei, alle XML-Tags aus dem Dokument zu entfernen, bevor man es indexiert. Der Nachteil daran ist allerdings, dass man jegliche Information, die in der Auszeichnung enthalten war, verliert. Eine andere Variante ist, die XML-Tags als Index-Terme zu behandeln und sie einfach mit in den Index aufzunehmen. Hier hat man nun die Möglichkeit, die Tags ohne die spitzen Klammern zu indexieren oder, um den Inhalt und die Struktur des Dokumentes im Index zu trennen, die Tags als ganzes zum Index hinzuzufügen. Letzteres hat den Vorteil, dass dann auch explizit nach dem Auftreten einzelner Elemente gesucht werden kann.

Das Indexieren als flache Dateien hat allerdings den Nachteil, dass Suchanfragen als Ergebnis nur das XML-Dokument als Ganzes zurückliefern können.

Semistrukturierte Indexierung

Bei der semistrukturierten Indexierung werden nicht alle strukturellen Informationen des Dokumentes zur Indexierung herangezogen.

Die *feldbasierte Indexierung* erweitert die Dokumentrepräsentation um eine Menge verschiedener Felder. Beispiele könnten separate Felder für Autoreninformationen, den Titel oder den Inhalt des Dokumentes sein. Die Terme der Dokumente werden dann im Index mit den ihnen zugeordenten Feldern abgelegt und ermöglichen so die Suche nach Dokumenten mit speziellen Eigenschaften, zum Beispiel kann so nach Dokumenten mit einem bestimmten Wort im Titel gesucht werden, oder man kann Termen in bestimmten Feldern ein höheres Gewicht zuordnen. Besonders leicht lässt sich diese Methode dann anwenden, wenn die erforderlichen Informationen bereits als Metadaten vorhanden sind, zum Beispiel als RDF oder innerhalb spezieller XML-Tags im Dokument.

Weitere Möglichkeiten für die semistrukturierte Indexierung sind die *segmentbasierte Indexierung* und die *baumbasierte Indexierung* (*tree-based indexing*), bei denen die Dokumente in Regionen bzw. Bäume zerlegt werden bevor sie im Index abgelegt werden.

Strukturierte Indexierung

Strukturierte Indexierung (*structured indexing*) berücksichtigt bei der Erzeugung des Index die komplette Struktur der XML-Dokumente. Oft ist es daher notwendig, dass alle Dokumente einer Kollektion auf derselben DTD basieren.

Ein Anwendungsgebiet der strukturierten Indexierung ist der Informationsaustausch zwischen Datenbanken und XML-Dokumenten (*IR/DB indexing*). Dazu müssen die Dokumente einer sehr strikten Struktur folgen, damit der Austausch beziehungsweise die Umwandlung zwischen Datenbank und XML-Dokumenten möglich ist. Beispiel 5 zeigt ein XML-Dokument und dessen Abbildung in einer Tabelle. Teilweise geht man sogar noch weiter und versucht, IR-Systeme zum Abfragen von Informationen aus einer Datenbank zu benutzen. Jede Zeile einer Datenbank wird vom IR-System als ein Dokument mit mehreren Feldern interpretiert, die beim Retrieval auf eine beliebige Art verwendet werden können. Ein Vorteil, den man gegenüber konventionellen Abfrage-sprachen für Datenbanken hat, ist, dass man eine Bewertung der Relevanz der Ergebnisse erhält.

Beispiel 5 **XML-Dokument und Abbildung in der Tabelle “student” nach [LLD+02]**

```
<?xml version=”1.0” ?>
<student>
<number>945182</number>
<name>Gerard Salton</name>
<contact>123456</contact>
</student>
```

number	name	contact
...	...	...
945182	Gerard Salton	123456
...	...	...

Im Kontext dieser Arbeit sehr wichtig ist die *pfadbasierte Indexierung* (*path-based indexing*). Ein Element innerhalb eines XML-Dokumentes lässt sich eindeutig über seinen XPath identifizieren. Deshalb wird bei der Indexierung für die Terme der XPath oder eine andere Repräsentation der Pfadinformationen zum zugehörigen Dokumentabschnitt mit im Index abgelegt, wobei als Terme nur der Dokumenteninhalt, nicht aber die XML-Tags verwendet werden. Dadurch bleiben die Informationen über den Aufbau des XML-Dokumentes erhalten. Manchmal werden auch zwei Indizes angelegt, wobei im zweiten Index die XPath-Ausdrücke so indiziert werden, als wären sie (unstrukturierte) Dokumente. Terme und XPath-Ausdrücke werden über einen Bezeichner (XID) miteinander verknüpft. Eine Anfrage kann dann aus einer Verknüpfung einer Anfrage an die XPath-Ausdrücke und an den Dokumenteninhalt bestehen. So kann man, ausreichendes Wissen über die im Dokument verwendete DTD vorausgesetzt, gezielt nach Termen innerhalb bestimmter Tags des Dokumentes suchen.

Ein weiterer Ansatz der strukturierten Indexierung ist das *positionsbasierte Indexieren*, wo das tatsächliche spätere Layout eines Dokumentes, das in Form von Styleinformationen zugänglich ist, dazu benutzt wird, die Struktur in einem Index abzuspeichern.

3.2.2 Suche

Die Suche innerhalb der XML-Dokumentenkollektion ist natürlich davon abhängig, welche Variante der Indexierung benutzt wurde. Selbstverständlich möchte man die gespeicherten Strukturinformationen und Metadaten zu Verbesserung der Ergebnisse einsetzen, aber man sollte auch in der Lage sein strukturelle Bedingungen auszuwerten, die möglicherweise in der Anfrage gestellt werden.

Die Indexierung mit flachen Dateien erlaubt dabei die wenigsten Möglichkeiten; je nach verwendeter Variante kann höchstens nach dem Auftreten spezieller Elemente innerhalb ganzer Dokumente gesucht werden.

Bei Dokumenten, bei denen der Index mit semistrukturierter Indizierung angelegt wurde, hat man schon mehr Optionen. Im Falle der feldbasierten Indizierung kann die Suche auf bestimmte Felder reduziert werden oder der Inhalt einzelner Felder wird bei der Berechnung der Relevanz bevorzugt.

Die strukturierte Indexierung gestattet die flexibelsten Möglichkeiten beim Retrieval. Die Suche besteht meist aus mehreren Phasen, die aber nicht unbedingt in einer strikten und abgeschlossenen Reihenfolge stattfinden. Zunächst wird der Index nach den Termen aus der Anfrage durchsucht. Dann werden für relevante Terme deren strukturelle Informationen abgefragt, um damit weitere Operationen auszuführen, das heißt sie in die Berechnung der Relevanz einzubeziehen. Die Kombination der Termgewichte mit den strukturellen Bedingungen kann an dieser Stelle gegebenenfalls noch hinzukommen. Die letzte Phase betrachtet alle Ergebnisse desselben Dokuments, die dann zum endgültigen Ergebnis kombiniert oder ausgewählt werden.

3.3 Bestimmung der Ergebnisse

Die Berechnung der Ergebnisse für die Suche auf XML-Dokumenten gestaltet sich komplizierter als beim klassischen Retrieval auf unstrukturierten Elementen. Zwei zusätzliche Aspekte gilt es hier zu beachten.

- Wie wirkt sich das Wissen über die Struktur der Dokumente auf die Berechnung der Relevanz aus?
- In welcher Granularität sollen die Ergebnisse zurückgeliefert werden?

3.3.1 Relevanzberechnung

Da sich auf dem Gebiet des XML-Retrieval noch wenige Standardlösungen herauskristallisiert haben, werden hier einige Überlegungen herausgestellt, die verdeutlichen sollen, dass durch die Struktur der XML-Dokumente und die verschiedenen Methoden des Indexierens bei der Relevanzberechnung völlig neue Aspekte betrachtet werden müssen.

Bei der Berechnung der Relevanz wurden in Abschnitt 2.5 die Termgewichte und im Zusammenhang damit die inverse Dokumenthäufigkeit (IDF) eingeführt. Da sich die IDF aus dem Auftreten eines Termes innerhalb der Dokumentenkollektion berechnet, Ergebnisse beim XML-Retrieval aber auch in Form von Dokumentfragmenten vorliegen können, muss man unter Umständen mit einer anderen Maßzahl rechnen. Mass und Mandelbrod [MaMa03] verwenden stattdessen eine Fragmenthäufigkeit CF(t) (*component frequency*), die sich aus der Vorkommenshäufigkeit eines Termes t in Dokumentfragmenten berechnet, und eine Termhäufigkeit $TF_C(t)$ basierend auf der Anzahl des Auftretens eines Terms t innerhalb eines Fragments C (siehe Beispiel 6).

Damit können auf dem so vorbereiteten Index zum Beispiel Abwandlungen klassischer IR-Methoden angewandt werden. Resultierend erhält man Informationen über die Relevanz einzelner Dokumentfragmente, die nun noch weiter mit dem Wissen über die Dokumentstruktur kombiniert werden können. Dazu gibt es bereits mehrere Ansätze, wie beispielsweise die Idee von Frisse [Fris87]. Er berechnet die Relevanz für ein Dokumentfragment und kombiniert das Ergebnis mit den Relevanzbewertungen der Kindelemente im Dokumentbaum, geteilt durch deren Anzahl. Eine andere Möglichkeit besteht darin, den Inhalt abhängig von der Auszeichnung in verschiedene Datentypen mit unterschiedlicher Gewichtung einzuteilen.

Beispiel 6 **Berechnung der Fragmenthäufigkeit [MM03]**

In diesem Beispiel bestehe die gesamte Dokumentenkollektion aus einem einzigen Dokument mit drei Komponenten und zwei Termen:

```
<article>
    t1
    <sec>
        <p>
        t2
        </p>
    </sec>
</article>
```

Damit ergibt sich:

$CF(t_1) = 1$, $CF(t_2) = 3$;
$TF_{article}(t_1) = 1$, $TF_{sec}(t_1) = 0$, $TF_p(t1) = 0$;
$TF_{article}(t_2) = 1$, $TF_{sec}(t_2) = 1$, $TF_p(t_1) = 1$.

Die jeweiligen Ansätze sind natürlich stark vom verwendeten Retrievalmodell abhängig. Eine Übersicht der Anpassungen für verschiedene Retrievalmodelle geben Luk et al. [LLD+02]. Abschnitt 3.5 zeigt exemplarisch zwei aktuelle Forschungsansätze.

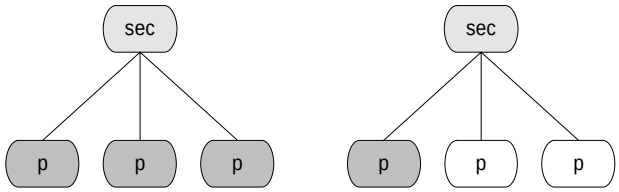
3.3.2 Granularität der Ergebnisse

Ein großer Vorteil beim Retrieval auf XML-Dokumenten, ist die Möglichkeit, nicht nur das gesamte Dokument zurückzuliefern, sondern auch einzelne Teile davon. Das ist beispielsweise dann wünschenswert, wenn in einem langen Dokument nur ein Absatz oder ein Kapitel für den Informationsbedarf des Benutzers relevant ist.

Wenn die Suchergebnisse XML-Dokumentfragmente enthalten können, ergibt sich zunächst die Frage nach der Größe der Fragmente. Sind die Fragmente zu klein, dann könnte es sein, dass dem Benutzer der nötige Kontext zum Verständnis fehlt; sind die Fragmente hingegen zu groß, ist unter Umständen noch sehr viel irrelevante Information darin enthalten.

Ein weiteres Problem ergibt sich, wenn sich die Ergebnisse überlappen, das heißt wenn sowohl ein Dokumentfragment als auch darin eingebettete Fragmente relevant sind. In Abbildung 7 links seien alle drei Absätze („*p*“) innerhalb eines Abschnittes („*sec*“) relevant; in Abbildung 7 rechts sei nur der erste Absatz relevant. Normalerweise würde ein IR-System für den ersten Fall alle Absätze und den Abschnitt als Ergebnis zurückliefern. Bei dem zweiten Fall würde der erste Absatz und der Abschnitt zurückgeliefert.

Abbildung 7 Überlappende Ergebnisse; die Einfärbung gibt den Grad an Relevanz an



Es ist leicht einzusehen, dass der Benutzer damit mit vielen redundanten Ergebnissen konfrontiert wird, die er alle durchsehen muss. Ein wünschenswertes Verhalten wäre daher, wenn das IR-System für den ersten Fall nur den Abschnitt und nicht die Absätze zurückliefert, für den zweiten Fall nur den ersten Absatz.

3.4 Evaluierung und INEX

Die *Initiative for the Evaluation of XML Retrieval* wurde Anfang 2002 gegründet mit dem Ziel, ein Forum für das Gebiet des XML-Retrieval zu sein, mit dem sich Forschungsergebnisse miteinander vergleichen lassen. Zu diesem Zweck bietet INEX eine Testkollektion von XML-Dokumenten und einheitliche Bewertungsmaßstäbe zum Vergleich der Ergebnisse. Auf dem jährlich

stattfindenden INEX-Workshop werden die jeweils aktuellsten Forschungsergebnisse diskutiert, mit Schwerpunkt auf dem Ad-Hoc-Retrieval von XML-Dokumenten.

3.4.1 Die INEX-Testkollektion

Die INEX-Testkollektion besteht aus einer Menge von Dokumenten, einer Menge von ausführlichen Beschreibungen bestimmter Informationsbedarfe (*Topics*) und Bewertungen über deren Relevanz.

Die Dokumente wurden der INEX von der IEEE überlassen. Es handelt sich dabei um den Volltext von über 12000 Artikeln aus Veröffentlichungen der IEEE. Alle Dokumente sind mit durchschnittlich 1532 XML-Elementen pro Artikel ausgezeichnet, bei einer durchschnittlichen Verschachtelungstiefe von 6,9 [FuLa04].

Die INEX-Topics wurden von den an der Konferenz teilnehmenden Forschungsgruppen erstellt. Sie bestehen hier aus dem Titel des Topics (*Title*), einer kurzen Beschreibung (*Description*), einer ausführlicheren Beschreibung (*Narrative*) und den Schlüsselwörtern (*Keywords*). Sie werden in die zwei Gruppen CO und CAS eingeteilt. Die meisten Forschungsansätze unterstützen beide Gruppen, häufig unterscheidet sich jedoch das Vorgehen bei der Verarbeitung.

CO - Content Only

CO-Anfragen (*Content Only*) berücksichtigen die Struktur der Dokumente in der Anfrage nicht und enthalten nur inhaltsbezogene Elemente. Das Retrieval-System soll die Elemente zurückliefern, die für die Anfrage relevant sind, ohne genauere Angaben über die Granularität oder Ähnliches zu haben. Die Idee hinter den CO-Topics ist die Tatsache, dass viele Benutzer entweder keine genaueren Informationen über den Aufbau der Dokumente haben oder sich nicht damit beschäftigen wollen. Das IR-System muss daher versuchen, selbständig die relevanten Elemente zu identifizieren.

CAS - Content And Structure

CAS-Anfragen (*Content And Structure*) erlauben die Einschränkung der Suche auf bestimmte strukturelle Elemente innerhalb der XML-Dokumente.

Bei der INEX 2004 wird eine Abwandlung benutzt, die sogenannten VCAS-Topics (*Vague Content And Structure*). Diese Anfragen enthalten spezifische Angaben über den Kontext der gewünschten Ergebnisse, also z.B. das Vorhandensein spezieller XML-Tags. Da aber für einen Benutzer die exakte Spezifizierung solcher struktureller Bedingungen relativ kompliziert und fehleranfällig ist, wird bei den VCAS-Topics davon ausgegangen, dass diese Bedingungen eine ähnliche Unschärfe aufweisen wie der Anfrageteil für den Inhalt. Das IR-System sollte den Teil der Anfrage, der sich auf die Struktur bezieht eher als Hinweis darauf betrachten, wo gesucht werden soll [LaMa04].

Die INEX-Konferenz 2003 definierte außerdem noch SCAS-Topics (*Strict Content And Structure*), bei denen die in der Anfrage angegebenen strukturellen Bedingungen von den Ergebnissen exakt erfüllt werden mussten. Diese wurden jedoch wieder fallengelassen, vermutlich weil sie eine zu starke Beschränkung des Retrieval bedeuten. Vor allem aus einer anwendungsbezogenen Perspektive ist dies nicht sinnvoll, da dabei ein sehr detailliertes Kontextwissen der Benutzer vorausgesetzt wird.

3.4.2 Bewertungsmaßstäbe für XML-Dokumentfragmente

Die zusätzlichen Informationen innerhalb der XML-Dokumente und vor allem die Möglichkeit, auch Dokumentfragmente als Ergebnis zurückzuliefern, haben auch Auswirkungen auf die Bewertung der Relevanz. Beim traditionellen Retrieval werden einige implizite Annahmen über die Dokumente der zugrundeliegenden Kollektion getroffen, die sich beim Retrieval auf XML-Dokumenten nicht ohne weiteres halten lassen [GKFL03]:

- Die meisten Retrieval-Modelle verlassen sich darauf, dass Dokumente voneinander unabhängige Einheiten sind. Da beim XML-Retrieval auch Teile von Dokumenten als Ergebnis erlaubt sind, können unterschiedliche Teile desselben Dokuments kaum als voneinander unabhängig angesehen werden.

- Beim traditionellen Retrieval werden die Resultate als separate, eindeutig unterscheidbare Einheiten interpretiert, da nur vollständige Dokumente im Ergebnis enthalten sind. Das XML-Retrieval gestattet es jedoch beispielsweise, einen Abschnitt und einen darin enthaltenen Absatz als zwei verschiedene Ergebnisse zu präsentieren. Die Überlappung dieser Ergebnisse sollte dann natürlich in die Bewertung einfliessen.

- Normalerweise geht man davon aus, dass die Dokumente ungefähr denselben Umfang haben, das heißt, dass ein Benutzer zur Feststellung, ob ein Ergebnis für ihn relevant ist, für jedes Ergebnis in etwa dieselbe Zeit benötigt. Da aber die durch das XML-Retrieval erzeugten Ergebnisse je nachdem, ob ein Ergebnis ein ganzer Artikel oder nur ein kurzer Abschnitt ist, eine sehr unterschiedliche Größe aufweisen, sollte auch dieser Faktor bei der Bewertung berücksichtigt werden.

- Bei einer Ergebnisliste, bei der die Ergebnisse nach Relevanz sortiert sind, wird ein Benutzer diese Ergebnisse eines nach dem anderen durchgehen. Im traditionellen Retrieval ist das unproblematisch, beim XML-Retrieval kann diese Liste aber durchaus mehrere Ergebnisse desselben Dokuments enthalten. Um dem Benutzer nicht zu viele Kontextwechsel zuzumuten, kann es unter Umständen sinnvoll sein, alle Ergebnisse für ein Dokument nacheinander zu präsentieren.

INEX berücksichtigt diese Punkte durch verschiedene Bewertungmaßstäbe speziell für das Information Retrieval auf XML-Dokumenten. Zunächst wird die Definition von Relevanz auf zwei Dimensionen erweitert:

- *Exhaustivity* (Gründlichkeit, Vollständigkeit) beschreibt, wie erschöpfend ein Dokumentfragment das Thema eines Topics behandelt;

- *Specificity* (Genauigkeit, Spezifität) beschreibt, wie sehr sich das Dokumentfragment auf das Thema des Topics konzentriert, ohne für den Benutzer irrelevante Informationen zu enthalten.

Exhaustivity wird kurz als *e-Wert*, Specificity als *s-Wert* bezeichnet. Für beide Dimensionen wurden vier Abstufungen definiert, die jeweils von 0 (*not exhaustive* beziehungsweise *not specific*) bis 3 (*highly exhaustive* beziehungsweise *highly specific*) reichen. Die Relevanz der Dokumentfragmente wird mit diesen beiden Werten beschrieben, wobei es möglich ist, dass ein Fragment einen hohen e-Wert, aber nur einen niedrigen s-Wert erhält und umgekehrt. Nur wenn das Fragment ein Thema überhaupt nicht behandelt, was einem e-Wert von 0 entspricht, muss folglich auch der s-Wert 0 sein [KaLP04].

Um mit den traditionellen Bewertungsverfahren Recall und Precision (siehe Abschnitt 2.6.2) arbeiten zu können, müssen die Werte für Exhaustivity und Specificity durch eine Quantisierungsfunktion auf einen einzelnen Relevanzwert gebracht werden. Dazu definiert INEX die zwei Funktionen f_{strict} und f_{gen} [KaLV04]. Mit der Funktion f_{strict} wird bewertet, ob eine Retrievalmethode in der Lage ist, hochgradig vollständige und genaue Ergebnisse zu liefern:

$$f_{strict}(e, s) = \begin{cases} 1 & \text{wenn e = 3 und s = 3,} \\ 0 & \text{sonst.} \end{cases}$$

Mit der Funktion f_{gen} können Dokumente nach ihrem Grad an Relevanz bewertet werden:

$$f_{gen}(e, s) = \begin{cases} 1 & \text{wenn } (e, s) = (3, 3) \\ 0.75 & \text{wenn } ((e, s) \in \{(2, 3), (3, \{2, 1\})\}) \\ 0.5 & \text{wenn } ((e, s) \in \{(1, 3), (2, \{2, 1\})\}) \\ 0.25 & \text{wenn } ((e, s) \in \{(1, 2), (1, 1)\}) \\ 0 & \text{wenn } (e, s) = (0, 0) \end{cases}$$

Basierend auf der Quantifizierung können nun Standardverfahren für die Berechnung von Recall-Precision-Diagrammen angewandt werden.

Ein Problem dabei ist allerdings, dass überlappende Ergebnisse nicht berücksichtigt werden, was dazu führen kann, dass Systeme, die vermehrt diese verschachtelten Ergebnisse zurückliefern, besser bewertet werden. Zwar ist nicht ganz klar, wie man diese Problematik bei der Evaluation der Ergebnisse zufriedenstellend lösen kann, eindeutig ist jedoch, dass überlappende Ergebnisse sich negativ auf die Bewertung der Effektivität auswirken sollten. Eine Vergleichsmöglichkeit bietet der *Overlap Indicator* [VrKL04], der eine Liste R von Ergebnissen nach dem prozentualen Anteil der überlappenden Ergebnisse darin bewertet:

$$\frac{|\{p \in R | (\exists (q \in R) \wedge p \neq q \wedge \text{overlap}(p, q))\}|}{|R|}$$

Der Overlap Indicator wird zusätzlich zu den Recall- und Precision-Werten angegeben und fließt nicht in die Bewertung ein.

Es besteht allerdings auch die Möglichkeit, sowohl die Größe der Fragmente als auch das Maß der Überlappung in die Berechnung von Recall und Precision einzubeziehen. Gövert et al. [GKFL03] berechnen Werte für Recall und Precision nicht wie in Abschnitt 2.6.2 nach einer bestimmten Anzahl Ergebnisse, sondern verwenden die Größe der Fragmente als Parameter. Die Überlappung wird dadurch miteinberechnet, dass für ein Ergebnis die Werte für Recall und Precision mit dem Anteil des Fragments multipliziert werden, der neu ist, also noch in keinem anderen Ergebnis enthalten war.

Mit welchen Metriken XML-Retrievalsysteme am Besten bewertet werden sollten ist immer noch ein offenes Thema. Der INEX-Workshop 2004 sollte das Verhalten verschiedener Metriken untersuchen. Eine abschließende Bewertung der verschiedenen Metriken steht allerdings noch aus.

3.5 Ansätze für das XML Retrieval

Nach dem aktuellen Stand der Forschung ist es unklar, welche Retrieval-Modelle für die Suche auf XML-Dokumenten am Besten geeignet sind. Die Ergebnisse von INEX zeigen, dass in einigen Bereichen die bestehenden Ansätze immer noch ungenügend sind. Daher existieren auch sehr viele verschiedene Ansätze und Varianten. Dieser Abschnitt zeigt, wie sich diese Ansätze nach verschiedenen Kriterien kategorisieren lassen und stellt dann die beiden IR-Systeme HyREX und XXL vor.

3.5.1 Kategorisierungen

Die aktuellen Forschungsansätze für das XML-Retrieval lassen sich unter unterschiedlichen Gesichtspunkten betrachten. In diesem Abschnitt führen wir eine Kategorisierung in mehreren Dimensionen ein, die es uns ermöglicht, unterschiedliche Systeme zu vergleichen und einzuordnen.

Kategorisierung nach dem Ansatz

Die Basis für unsere Kategorisierung ist die Einteilung dem *Ansatz* der Systeme:

- *IR-Modell-orientierte Ansätze* beschäftigen sich mit der Erweiterung bestimmter Arten von etablierten IR-Modellen, zum Beispiel dem Vektorraummodell oder dem wahrscheinlichkeitstheoretischen Retrieval, zur Verwendung mit XML-Dokumenten. Das im IBM Research Lab in Haifa entwickelte System JuruXML [MMA+02] ist ein Beispiel für einen IR-Modell-orientierten Ansatz, der das Vektorraummodell an die Anforderungen von XML-Dokumenten anzupassen versucht.
- *DB-orientierte* Ansätze erweitern Datenbanksysteme dahingehend, dass sie semistrukturierte Daten verwenden können. Anders als normale DB-Systeme erzeugen die meisten dieser Ansätze, wie zum Beispiel das CIRQUID Projekt [Hiem02], eine Bewertung der Resultate.
- *XML-spezifische Ansätze* bauen nicht auf existierenden Ansätzen auf, sondern wurden speziell für die Arbeit mit XML-Dokumenten entwickelt. Meist bauen diese Ansätze allerdings auf XML-Standards wie XPath oder XQuery auf.

Oft lassen sich die Ansätze jedoch nicht eindeutig einer dieser Kategorien zuordnen, sondern sie integrieren mehrere Aspekte. Ein Beispiel für einen sowohl IR-Modell- als auch DB-orientierten Ansatz ist der von Pehcevski, Thom und Vercoustre [PeTV03], die eine XML-Datenbank mit einem IR-System für unstrukturierte Dokumente kombinieren.

Kategorisierung nach dem Anwendungsgebiet

Die Standardfunktionalität aller Systeme ist das *Ad-Hoc-Retrieval*. Unter Ad-Hoc-Retrieval versteht man in diesem Zusammenhang die Suche so, wie man ein klassisches Bibliothekssystem benutzen würde mit dem Unterschied, dass sowohl Anfragen an die Struktur der Dokumente als auch das Zurückgeben von Dokumentfragmenten möglich sind. Als Basis wird dabei eine festgelegte, einheitliche Dokumentenkollektion verwendet.

Einige Systeme implementieren zusätzlich eine Form von *Relevanz-Feedback* (siehe Abschnitt 2.5.2). Ein Benutzer erhält also die Möglichkeit, die Dokumentfragmente der Ergebnismenge als relevant oder irrelevant für seinen Informationsbedarf zu markieren und das System

benutzt diese zusätzlichen Informationen zum Verbessern der Ergebnisse. Im Unterschied zu klassischen IR-Systemen, die Relevanz-Feedback implementieren, sollten dann auch die strukturellen Informationen der XML-Dokumente für die Bearbeitung des Relevanz-Feedback verwendet werden. Mass und Mandelbrod [MaMa04] zeigen, wie sich bestehende Relevanz-Feedback-Algorithmen für die Verwendung mit XML-Dokumentfragmenten erweitern lassen.

Eine andere Form der Erweiterung ist die Ausdehnung des Retrieval auf *heterogene Dokumentenkollektionen*. Hier besteht die Dokumentenkollektion aus XML-Dokumenten aus unterschiedlichen Quellen mit unterschiedlichen und möglicherweise nicht im voraus bekannten DTDs. Diese Form des Retrieval verlangt nach anderen Ansätzen, da sich die meisten Systeme sehr stark darauf konzentrieren, Informationen aus der DTD der Kollektion beim Retrieval zu verwenden, um zum Beispiel die Elemente, die eine Antwort repräsentieren können zu definieren. Weitere Probleme entstehen bei Content-And-Structure-Anfragen, wo die Bedingungen aus der Anfrage auf die – in diesem Fall unbekannte – Dokumentstruktur abgebildet werden müssen. Auf diesem recht neuen Gebiet gibt es noch relativ wenige Erkenntnisse. Das EXTIRP-2004-Projekt [Leht04] wurde zwar mit einer homogenen Dokumentenkollektion getestet, verwendet aber keinerlei aus der DTD abgeleiteten Informationen und kann daher als Forschungsansatz für heterogene Dokumentenkollektionen klassifiziert werden.

Andere Aspekte

Darüberhinaus existieren noch einige andere Forschungsgebiete, die sich nicht ohne weiteres in die oben eingeführten Kategorien einordenen lassen und hier nur beispielhaft erwähnt werden, da sich das Gebiet des XML-Retrieval in immer neue Richtungen bewegt.

- Das NLPX-System [WoGe04] hat das Ziel, sowohl inhaltliche als auch strukturelle Bedingungen zu interpretieren, die in natürlicher Sprache an das System formuliert werden.
- Der Ansatz von Kelly, Geva, Sahama und Loke [KGSL03] beschreibt die Implementierung einer verteilten XML-Suchmaschine, bei der ein zentraler Server die Suchanfragen bearbeitet und die Dokumente und Indizes sich verteilt auf verschiedenen Clients befinden.
- Kim und Son [KiSo04] untersuchten das Verhalten von Benutzern, die mit XML-Dokumentfragmenten als Suchergebnisse konfrontiert wurden. Ziel war es anhand verschiedener Faktoren, wie der Anzahl der Suchbegriffe, der Anzahl der Ergebnisse oder der benötigten Zeit, herauszufinden, welche davon Einfluss auf die Zufriedenheit des Benutzers haben.

Im Folgenden werden zwei praxiserprobte Systeme etwas detaillierter besprochen. Das Kapitel schließt mit einem Vergleich der beiden Ansätze.

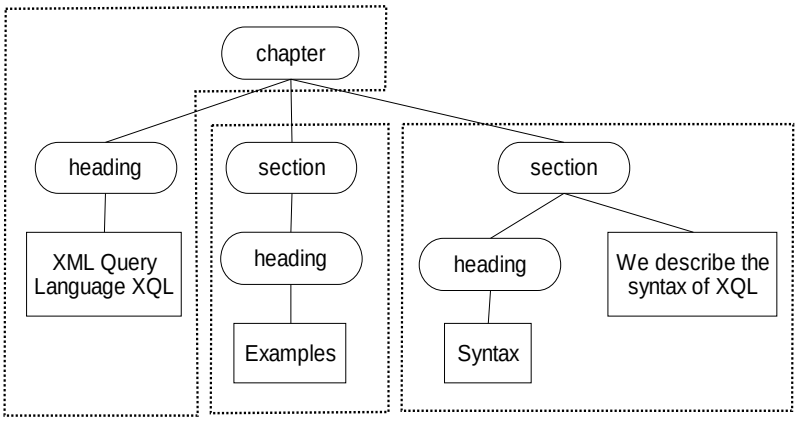
3.5.2 HyREX

Das HyREX-System (*Hyper-media Retrieval Engine for XML*) wurde von Fuhr und Abolhassani an der Universität Duisburg entwickelt ([FuGr01], [GAFG02]). Es handelt sich dabei um einen IR-Modellorientierten Ansatz, basierend auf dem wahrscheinlichkeitstheoretischen Modell. HyREX unterstützt sowohl CO- als auch CAS-Anfragen für das Ad-Hoc-Retrieval.

Für die Termgewichtung verwendet HyREX das Konzept der Indexknoten. Bestimmte Elemente der Auszeichnung werden als Indexknoten festgelegt und erhalten eine eindeutige ID, in Beispiel 7 sind dies die Elemente „*chapter*“ und „*section*“. Der Text innerhalb dieser Elemente wird indexiert. HyREX geht davon aus, dass die Indexknoten zwar disjunkt sein sollten, trotzdem

sollte es möglich sein, Ergebnisse unterschiedlicher Granularität zu erhalten. Die Indexierung startet daher mit den feinkörnigsten Indexknoten, im Beispiel ist dies „*section*“. Bei Indexknoten auf höherer Ebene wird nur der Text indexiert, der nicht schon in bereits indexierten Knoten enthalten ist. Im Beispiel ist für den Indexknoten „*chapter*“ nur noch der Text innerhalb des Elements „*heading*“ zu indexieren, der Rest ist schon in den beiden „*section*“ Indexknoten enthalten.

Beispiel 7 **Baumdarstellung eines XML-Dokumentsfragments mit Indexknoten nach [FuGr01]**



Ist das Dokument auf diese Weise in eine Menge disjunkter Indexknoten zerlegt, werden darauf bekannte Termgewichtungsverfahren wie z.B. eine tf-idf-Formel angewandt. Damit entspricht das Auftreten eines Terms in einem Indexknoten einem probabilistischen Ereignis, das durch das Paar [*Knoten-ID, Term*] eindeutig identifizierbar ist.

HyREX geht beim Retrieval, je nachdem, ob strukturelle Bedingungen formuliert werden oder nicht, unterschiedlich vor. Zur Durchführung von Anfragen mit strukturellen Bedingungen verwendet HyREX die Anfragesprache XIRQL (*XML IR Query Language*). XIRQL erweitert eine Teilmenge des XQuery-Standards um IR-Methoden. Zur Berechnung der Relevanz werden zunächst die Termgewichte aus den Indexknoten ermittelt, die dann durch die strukturellen Bedingungen nochmal gefiltert werden. Eine Anfrage wird dabei in eine boolesche Kombination probabilistischer Ereignisse umgeformt.

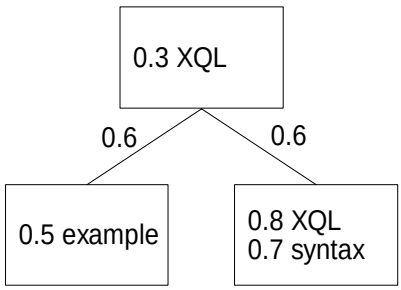
Für das folgende Beispiel nehmen wir an, dass der Indexknoten „*chapter*“ die ID 3, der rechte „*section*“-Indexknoten die ID 5 hat. Weiterhin wird angenommen, dass es ein Prädikat „cw“ gibt, das auf das Auftreten eines Wortes testet. Die Anfrage aus Beispiel 8 ergibt dann den Ausdruck $([3, \text{XQL}] \vee [5, \text{XQL}]) \wedge [5, \text{syntax}]$. Nach einigen weiteren Umformungen wird die Wahrscheinlichkeit mit P([3, XQL]) · P([3, XQL]) + P([3, XQL]) · P([3, XQL]) - P([3, XQL]) · P([3, XQL]) · P([3, XQL]) berechnet. Es resultiert eine Menge von Ergebnissen, die den Bedingungen genügt und nach Relevanz sortiert ist.

Beispiel 8 **XIRQL-Beispielanfrage für strukturelle Bedingungen in HyREX**

```
//chapter[.//* $cw$ "XQL" $and$ .//* $cw$ "syntax"]
```

Für eine einfache Suche ohne strukturelle Bedingungen werden Gewichte für alle Indexknoten berechnet, wobei für Indexknoten auf höherer Ebene dessen Termgewichte mit den Gewichten darin enthaltener Indexknoten kombiniert werden müssen. Um möglichst immer das spezifischste Element zu erhalten, werden die Termgewichte herabgewertet, wenn sie in die Bewertung eines Indexknotena auf höherer Ebene einfließen sollen. Beispiel 9 zeigt Gewichtungen für Terme des Dokumentes aus Beispiel 7. Die Zahlen vor den Termen sind die Gewichtungen der Terme vor der Abwertung. Die Zahlen an den Linien kennzeichnen den sogenannten Augmentierungsfaktor. Dieser Augmentierungsfaktor wird als wahrscheinlichkeitstheoretisches Ereignis an den betreffenden Indexknoten angehängt. Eine Suche nach „XQL“ würde für den *„chapter“*-Knoten folgenden Ereignisausdruck liefern: $[3, \text{XQL}] \vee [5] \wedge [5, \text{XQL}]$. Daraus errechnet sich dann mit P([5]) = 0.6 eine Wahrscheinlichkeit von 0.636 für den *„chapter“*-Knoten.

Beispiel 9 **Berechnung der Termgewichte ohne strukturelle Bedingungen**



Zusätzlich unterstützt XIRQL Datentypen mit vagen Prädikaten. Zur Verdeutlichung des Begriffs „vage Prädikate“ gibt Fuhr in [FuGr01] das Beispiel der Anfrage *„Suche Informationen über das Werk eines Künstlers namens Ulbrich, der um 1900 im Rhein-Main-Gebiet tätig war*“, wo das IR-System in der Lage sein soll, einen Artikel über Ernst Olbrichs Arbeit in Darmstadt 1899 zu finden. Es wird eine erweiterbare Typhierarchie verwendet mit vagen Prädikaten für jeden Datentyp. Als Beispiel für eine Typhierarchie führt Fuhr die Hierarchie *Text – westliche Sprache – deutscher Text* an. Auf der Textebene ist nur ein Substring-Match möglich, auf der Ebene westliche Sprache kann schon mittels Wortsuche, Trunkierung und Wortabstandssuche gearbeitet werden. Die unterste Ebene deutscher Text erlaubt schließlich Stammformensuche oder die Suche nach Teilen zusammengesetzter Wörter.

Ein weiterer Aspekt von XIRQL ist der sogenannte semantische oder strukturelle Relativismus. Darunter versteht man das Fallenlassen der Unterscheidung zwischen Attribut und Element, die Generalisierung von Datentypen beziehungsweise das Ausnutzen von Ontologien auf Elementnamen.

3.5.3 XXL

Das von Weikum, Theobald und Schenkel am Max-Planck Institut für Informatik in Saarbrücken entwickelte XXL-System ([ThWe02], [ScTW04]) fokussiert noch stärker auf den Aspekt der Ontologien und macht sie zu einem zentralen Punkt des Systems. Auch XXL ist ein IR-Modell-orientierter Ansatz für das Ad-Hoc-Retrieval.

Basierend auf Anfragesprachen wie XQuery wurde XXL (Flexible XML Search Language) entwickelt, mit einer Syntax ähnlich SQL (siehe Beispiel 10).

Beispiel 10 **XXL-Anfrage**

```
SELECT $T
FROM INDEX
WHERE ~article AS $A
AND $A/~title AS $T
AND $A/#/~section ~ “star | planet”
```

Besonders hervorzuheben ist hier der Operator „~“, mit dem man nach semantischen Ähnlichkeiten sowohl in den Element- und Attributnamen als auch in deren Inhalt suchen kann.

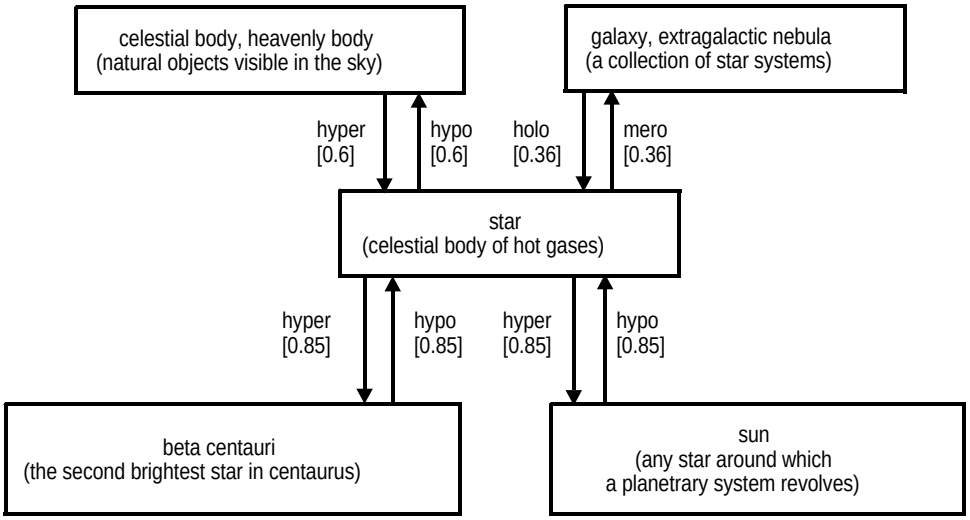
Die Suche basiert wieder auf einer Baumdarstellung von XML-Dokumenten, die hier zu einem gerichteten Graph erweitert wird, indem XXL auch Referenzen zwischen verschiedenen Dokumenten berücksichtigt. Eine Anfrage wie die aus Beispiel 10 wird zunächst in verschiedene Teilanfragen zerlegt, die als nicht-deterministischer endlicher Automat repräsentiert werden. Solche Teilanfragen werden nochmals in einzelne Ausdrücke zerlegt, die dann mit Hilfe des Index ausgewertet werden. Abschließend werden die Teilergebnisse wieder zu einem Ergebnis für die ursprüngliche Anfrage kombiniert.

Zur Berechnung von Teilausdrücken, die den Operator „~“ verwenden, wird ein sogenannter Ontologie-Graph verwendet. Beispiel 11 zeigt einen Ausschnitt eines solchen Graph. Dieser Graph besitzt jeweils ein semantisches Konzept als Knoten und deren Beziehungen als gewichtete Kanten.

Mögliche Beziehungen sind zwischen übergeordneten Begriffen, *hyper* beziehungsweise *hypo* im Beispiel, oder zwischen einem Begriff, der Teil eines anderen ist, im Beispiel als *holo* beziehungsweise *mero* gekennzeichnet. Ausserdem sind Synonyme im Graph gespeichert. Für zwei beliebige Knoten ist die Ähnlichkeit der Knoten definiert als das Produkt der Gewichtungen der Kanten auf dem Pfad zwischen diesen beiden Knoten.

Wird nun der Operator „~“ in der Form „Element ~ Term“ benutzt, so werden in einem ersten Schritt durch eine Suche im Ontologie-Graph ähnliche Terme gesucht, die einen Relevanzwert π_1 erhalten. Der nächste Schritt berechnet für diese Terme einen Relevanzwert π_2 nach Standardverfahren mittels einer tf-idf-Formel. Durch Multiplikation wird daraus der endgültige Relevanzwert $\pi_1 \cdot \pi_2$ gebildet.

Beispiel 11 Ausschnitt aus einem Ontologie-Graph nach [ThWe02]



Das XXL-System verwendet drei verschiedene Indexstrukturen. Der *Element Path Index* enthält die notwendigen Daten, um Vater-Kind-Beziehungen zwischen Knoten oder andere strukturelle Informationen der XML-Dokumente abzufragen. Der *Element Content Index* speichert als invertierte Liste die Terme der Dokumente mit Informationen zu ihren Vorkommenshäufigkeiten. Der *Element Ontology Index* schließlich enthält den Ontologie-Graphen.

3.5.4 HyREX und XXL im Vergleich

Beide Systeme fallen in den Bereich des IR-Modell-orientierten Ad-Hoc-Retrieval basierend auf dem wahrscheinlichkeitstheoretischen Retrieval. Sie beherrschen sowohl CO-Anfragen als auch CAS-Anfragen. Während sich XXL sehr stark auf den Aspekt der Ontologien konzentriert, versucht HyREX viele verschiedene Ideen zu integrieren.

Das XXL-System schnitt beim INEX Workshop 2003 um einiges schlechter ab als HyREX, das sich immerhin in den meisten Kategorien unter den ersten 10 platzieren konnte. Eine mögliche Begründung für das schlechte Ergebnis von XXL ist die den Testdaten zugrunde liegende Struktur der Daten, die das Ausnutzen der Ontologien nicht gestatten.

Das wirft wiederum die Frage auf, ob die INEX-Testkollektion wirklich geeignet ist, unterschiedliche Forschungsansätze vergleichbar zu machen.

Kapitel 4

Konzept zur Implementierung einer XML-Volltext-Suchmaschine

Ziel dieser Arbeit war die Implementierung einer XML-Volltext-Suchmaschine. Die im letzten Kapitel beschriebenen Konzepte werden uns dabei als Grundlage dienen. Das Hauptaugenmerk liegt auf einem neuen Ansatz zur Verbesserung des Retrieval. Nach der im letzten Kapitel eingeführten Kategorisierung handelt es sich bei unserem XML-Retrieval-System um einen IR-Modell-orientierten Ansatz basierend auf einem vereinfachten Vektorraummodell. Das System unterstützt das Ad-Hoc-Retrieval auf heterogenen Dokumentenkollektionen mit CO- und einer Variante von CAS-Anfragen.

Die Dokumentenkollektion, die für diese Arbeit als Basis dient, sind die Linux-Howto-Dokumente des Linux-Documentation Project [TLDP05]. Die Dokumente sind mit DocBook [WaMu99] ausgezeichnet, einer XML-Sprache für das Erstellen technischer Dokumentationen im Computerumfeld. DocBook ist für diesen Ansatz besonders gut geeignet, weil es im Vergleich zu anderen XML-Sprachen einen sehr reichhaltigen Satz an Inline-Elementen bereithält.

In diesem Kapitel beschreiben wir die prinzipielle Funktionsweise von Indexierung und Suche unseres Ansatzes. Dazu werden wir zunächst das Kernkonzept, die Verwendung von Inline-Elementen zu Verbesserung des Retrieval, basierend auf den Ideen von [Dopi05], genauer beleuchten.

Dann gehen wir auf die Architektur der Implementierung ein. Sie wurde in Java durchgeführt und ist keine Implementierung einer Suchmaschine von Anfang an, sondern verwendet die Bibliothek Apache Lucene als Basis und erweitert sie dahingehend, dass damit sowohl XML-Dokumente indexiert und durchsucht werden können als auch das Konzept der Element Relationships, das wir im folgenden Abschnitt einführen, verwendet werden kann.

Da die Implementierung auf der Suchmaschinen-Bibliothek Apache Lucene [ApLu05] basiert, werden wir zunächst deren Fähigkeiten in Bezug auf Indexierung und Suche vorstellen. Dann beschreiben wir, wie wir Lucene zur Indexierung verwenden. Es folgt ein Kapitel über die Architektur der Suche und wie Element Relationships bei der Suche eingesetzt werden können. Anschließend gehen wir noch detailliert auf unser Konzept zur Zusammenfassung der Ergebnisse ein.

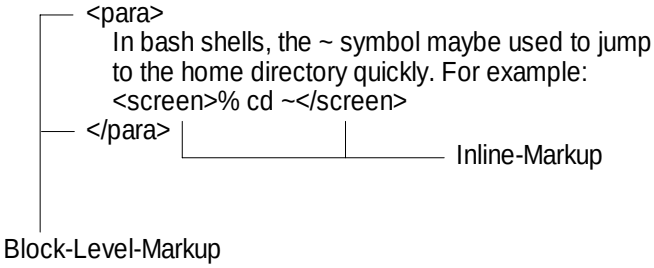
4.1 Verbesserung des Retrieval durch Element Relationships

Bevor wir näher auf unser Konzept eingehen, wie Inline-Markup mit einem *Element Relationship Graph* zur Verbesserung des XML-Retrieval eingesetzt werden kann und wie sich ein solcher Graph ableiten lässt, soll noch einmal der Unterschied zwischen Inline- und Block-Level-Markup verdeutlicht werden.

4.1.1 Inline-Markup und Block-Level-Markup

Im Zusammenhang mit dokumentenorientiertem XML wird zwischen Block-Level-Markup und Inline-Markup unterschieden. Mit Block-Level-Markup wird solches Markup bezeichnet, das das größere Abschnitte eines Textes in eine logische Struktur unterteilt. Im XML-Fragment aus Abbildung 8 ist das Element „*para*“ ein Block-Level-Element, das einen Absatz im Text markiert. Weitere Beispiele für Block-Level-Markup sind Elemente, die Abschnitte, Kapitel oder Blöcke von Metainformationen umschließen.

Abbildung 8 XML-Fragment mit Inline- und Block-Level-Markup (aus Deciding-Linux-HOWTO.xml)



Mit Inline-Markup werden Elemente bezeichnet, die innerhalb eines solchen Blocks bestimmte Teile eines Textes gesondert kennzeichnen. In Abbildung 8 markiert das Element „*screen*“, dass es sich beim Inhalt dieses Elementes um eine Bildschirm-Aus- oder -Eingabe handelt. Insbesondere ist es für Inline-Markup-Elemente erlaubt, dass sie als sogenannter *mixed content* vermischt mit Text in einem Element enthalten sein können.

Im Zusammenhang mit dem XML-Retrieval wird Block-Level-Markup meist dazu benutzt, gültige Ergebnisgranulate oder Einheiten für die Indexierung zu bestimmen. Abschnitt 4.5 befasst sich mit unserem Ansatz der Verwendung von Block-Level-Markup zur Zusammenfassung der Ergebnisse. Inline-Markup kann bei der Suche Verwendung finden, um den Inhalt eines solchen Elementes semantisch korrekt zu interpretieren. Auch bei unserem Ansatz soll Inline-Markup dazu genutzt werden, die Ergebnisse des Retrieval zu verbessern.

4.1.2 Problemstellung

Die Suche auf XML-Dokumenten liefert im Allgemeinen bessere Ergebnisse, wenn die Benutzer in der Lage sind, die Menge der Elemente einzugrenzen, in denen sinnvolle Ergebnisse zu erwar-

ten sind. In den meisten Fällen, wie zum Beispiel im Fall von XQuery-Full-Text (siehe Abschnitt 3.1.2) oder XIRQL (siehe Abschnitt 3.5.2) ist hierfür eine komplexe Anfragesprache vorgesehen. Man kann allerdings nicht davon ausgehen, dass Benutzer eines Retrievalsystems immer eine solche Sprache beherrschen oder willens sind, sie zu erlernen.

Doch selbst wenn ein Benutzer mit der Anfragesprache umgehen kann, können weitere Probleme entstehen. Insbesondere in XML-Sprachen, die eine besonders reichhaltige Auswahl an Inline-Elementen bieten, ist oft für die Benutzer nicht ganz eindeutig, welche Elemente des Markup den Informationsbedarf am Besten abbilden und daher in der Anfrage verwendet werden sollten. Auch für die Autoren können durch eventuelle Mehrdeutigkeiten des Markup Unklarheiten darüber entstehen, welches Element im konkreten Fall verwendet werden sollte.

Zusätzlich ist für das Retrieval zu beachten, dass XML-Dokumente oft zu dem Zweck ausgezeichnet werden, durch spätere Transformationen in ein Ausgabeformat wie HTML oder PDF die gewünschte optische Repräsentation zu erhalten. So kann es sein, dass die Autoren absichtlich nicht die semantisch korrekten Elemente verwenden, sondern viel allgemeinere. In der Dokumentation für DocBook [WaMu99] beispielsweise wird ein derartiges Problem erwähnt: „*Emphasis* wird oft dort verwendet, wo eine typographische Repräsentation erwünscht ist, auch wenn andere Markup-Elemente theoretisch sinnvoller wären.“ Dadurch können selbst bei korrekter Verwendung einer Anfragesprache nicht alle relevanten Ergebnisse gefunden werden.

4.1.3 Element Relationships

Um die angesprochenen Probleme zu behandeln, führen wir das Konzept der Element Relationships ein. Unser Ziel ist es dabei, eine Anfragesprache zu kreieren, die trotz ihrer Einfachheit eine erhebliche Verbesserung des Retrieval ermöglicht.

Wir bilden dafür aus der Menge der verfügbaren Inline-Elemente Kategorien verwandter Elemente, durch die wir Beziehungen zwischen diesen Elementen modellieren. Bei der Suche soll es dann möglich sein, eine dieser Kategorien zusätzlich zu den Suchtermen anzugeben. Ergebnisse, die in ein Element aus dieser Kategorie eingebettet sind, werden dann bei der Relevanzberechnung bevorzugt. Damit sich durch eine potenziell recht großen Anzahl von Kategorien die Anfragesprache nicht wieder verkompliziert, bilden wir eine Hierarchie von Kategorien, indem wir die Kategorien stufenweise verallgemeinern. Dadurch soll der Benutzer in die Lage versetzt werden, je nach seinem Kenntnisstand, präzisere Angaben durch Kategorien auf unteren Ebenen oder allgemeinere Angaben durch Kategorien auf höheren Ebenen zu machen.

Solche Kategorien müssen vorab für die der verwendeten Dokumentenkollektion zugrundeliegenden Schemadefinition erstellt werden. Damit ergibt sich natürlich die Frage, nach welchen Gesichtspunkten die Kategorien gebildet werden sollen. Dopichaj [Dopi05] schlägt dazu die folgenden Kriterien vor:

- **Tag-Namen.** Idealerweise sollten Elementnamen ihre Bedeutung ohne zusätzliche Information vermitteln. XXL (siehe Abschnitt 3.5.3) geht von dieser Annahme aus und verwendet Ontologien auf diese Namen. Allerdings scheint die Verwendung von sinnvollen Namen für XML-Tags eher die Ausnahme zu sein. Oft werden schwer zu verstehende Abkürzungen wie <quandadiv> aus DocBook oder
 aus XHTML verwendet.
- **Syntaktische Beschränkungen.** Mit XML-Schemadefinitionssprachen wie DTD oder XML Schema (siehe Abschnitt 3.1.1) kann die syntaktische Struktur der Dokumente vorgegeben

werden. Genauer gesagt kann die erlaubte Verschachtelung von Elementen definiert werden. In DocBook dürfen beispielsweise im Element <copyright> nur die Elemente <holder> und <year> enthalten sein. Diese Informationen können leicht aus den Schemadefinitionen abgeleitet werden, sind für uns jedoch nur von geringem Nutzen: Meist werden entweder alle Inline-Elemente erlaubt oder keines.

- **Semantik.** Da die Tag-Namen und die syntaktischen Beschränkungen allein nicht genügend Informationen enthalten, ist es notwendig, andere Informationen heranzuziehen, um semantische Beziehungen zwischen den Elementen herzustellen, zum Beispiel durch das Gruppieren verwandter Element-Typen. Diese Informationen sind normalerweise als Dokumentation für die Autoren der Dokumente zugänglich, jedoch kann das Zusammenstellen der relevanten Informationen aus diesen Dokumentationen sehr zeitaufwändig sein.
- **Inhalt.** Wenn eine genügend große Anzahl an Dokumenten vorhanden ist, können statistische Methoden, basierend auf dem Inhalt der XML-Elemente verwendet werden. Eine einfache Annäherung könnte statistische Informationen, wie zum Beispiel die Verwendung von Großbuchstaben beziehungsweise Kleinbuchstaben oder Zahlen, verwenden, um daraus Kategorien zu erzeugen. UNIX-Dateipfade enthalten beispielsweise eine im Vergleich zum Text hohe Anzahl von Schrägstrichen („/“). Weiterführende Ansätze könnten auch die Wörter innerhalb des Elementes in Beziehung zu deren Kontext dazu verwenden, Kategorien zu bestimmen.
- **Visuelle Darstellung.** Gewöhnlich wird dokumentenzentriertes XML dazu verwendet, eine visuelle Repräsentation für die Benutzer zu erstellen. Die Anzahl der Inline-Tags überschreitet meist die Anzahl der Formatierungsoptionen des Ausgabeformates, so dass viele unterschiedliche Tags gleich dargestellt werden. Dadurch gehen viele semantische Informationen aus dem Markup verloren, aber die Abbildung geschieht nicht willkürlich. Zwar werden diverse Tags, die in keiner semantischen Beziehung zueinander stehen, gleich abgebildet, verwandte Tags erhalten jedoch üblicherweise die gleiche Formatierung. In DocBook werden beispielsweise Computerbegriffe wie Dateinamen, Rechner-Ein- und -Ausgabe oder Namen von Umgebungsvariablen immer in einer nichtproportionalen Schrift dargestellt.

4.1.4 Element Relationship Graph

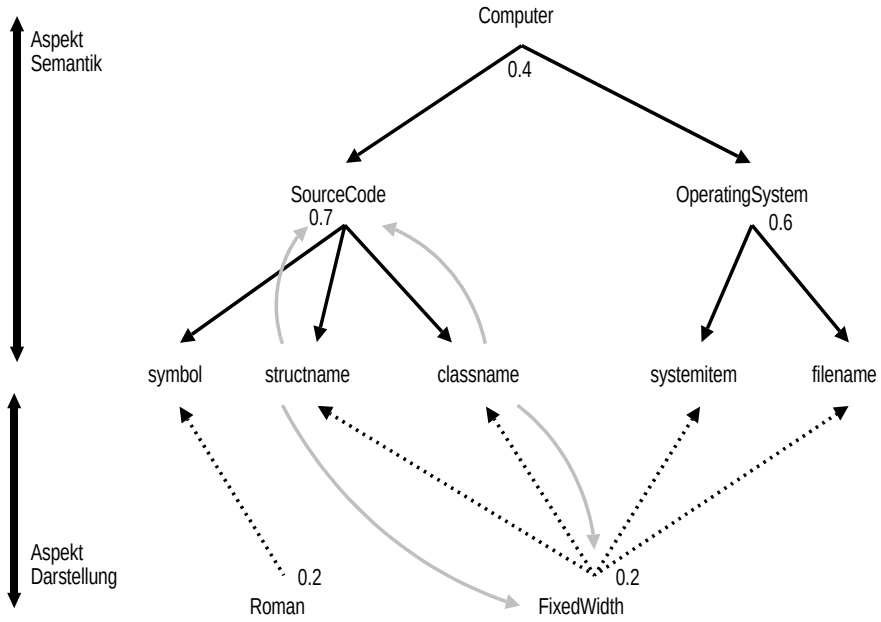
Die aus einem oder mehreren dieser Aspekte abgeleiteten Kategorien speichern wir in einer besonderen Struktur, dem *Element Relationship Graph* (ERG). Abbildung 9 zeigt einen Ausschnitt aus dem Graph, der für die Implementierung erstellt wurde.

Die Basis für den Graph sind die *Elementknoten*, die die Inline-Elemente repräsentieren. Darüber sind *Kategorieknoten* hierarchisch angeordnet. Man sieht Kategorien für die beiden Aspekte „Semantik“ und „Darstellung“. Ein ERG besteht also zunächst aus diversen Hierarchien für die einzelnen Aspekte, die durch die Elementknoten verbunden sind. Wir weichen die hierarchische Darstellung allerdings noch ein wenig auf und erlauben, dass sich Kategorien überlappen können.

Jeder Kategorieknoten n erhält nun einen Wert $c_n \in [0, 1]$, der die Kohärenz, also den logischen Zusammenhang seiner direkten Nachfolger angibt. Dadurch können wir berücksichtigen, dass unterschiedliche Aspekte mehr oder weniger Informationsgehalt haben. Beispielsweise ist die Information, dass die Elemente <structname> und <classname> beide mit demselben Schrifttyp dargestellt werden, nicht so aussagekräftig wie die Information, dass beide in der semantischen

Gruppe *SourceCode* sind. Die Kohärenzwerte müssen der folgenden Bedingung genügen, damit die Ähnlichkeit der Elemente bei einer Verallgemeinerung stets abnimmt: Wenn der innere Knoten a ein Vorfahre von d ist, dann muss gelten $c_a < c_d$.

Abbildung 9 Ausschnitt aus einem ERG für DocBook [Dopi05]



Um die Ähnlichkeit zweier unterschiedlicher Knoten n_1 und n_2 im ERG zu berechnen, müssen wir nun zunächst die Menge der nächstliegenden gemeinsamen Vorfahren A bestimmen. Die Ähnlichkeit der beiden Knoten berechnet sich dann als $max_{a \in A}(c_a)$.

Wir berechnen als Beispiel die Ähnlichkeit der beiden Knoten <structname> und <classname> anhand von Abbildung 9. Die Menge der nächstliegenden gemeinsamen Vorfahren für <structname> und <classname> ist {*SourceCode, FixedWidth*}. Damit resultiert eine Ähnlichkeit von $max(c_{SourceCode}; c_{FixedWidth}) = max(0{,}7; 0{,}2) = 0,7$.

Für die beiden Knoten <systemitem> und <classname> berechnen wir eine Kohärenz von $max(c_{Computer}; c_{FixedWidth}) = max(0{,}4; 0{,}2) = 0,4$. Damit wird auch der Unterschied deutlich zwischen unserem Ansatz und einem Ansatz, der die Ähnlichkeit als den kürzesten Pfad zwischen den Elementen betrachtet. Der nächstliegende gemeinsame Vorfahre basierend auf der Pfadlänge ist *FixedWidth*, aber *Computer* hat einen höheren Kohärenzwert und kann uns dadurch mehr Informationen über die Beziehung der beiden Elemente liefern.

Erstellung eines Element Relationship Graph

Für die Implementierung wurde ein Element Relationship Graph für DocBook mit den beiden Aspekten „Visuelle Darstellung“ und „Semantik“ erstellt.

Bei der Beschreibung der Aspekte wurde bereits erwähnt, dass die Erstellung eines ERG nur in manchen Fällen automatisiert werden kann. Soll der Graph auf der Semantik der Elemente basieren, muss er manuell erzeugt werden.

Eine mögliche semantische Gruppierung der Elemente wurde mit Hilfe der DocBook-Kurzreferenz erstellt. Leichte Modifikationen waren nötig, da die Autoren die Elemente mit der Betonung auf Übersichtlichkeit und nicht unbedingt semantische Ähnlichkeit gruppiert haben. Der Aspekt „Visuelle Darstellung“ wurde aus den offiziellen XSL Style Sheets, die für die Umwandlung von mit DocBook ausgezeichneten Dokumenten in HTML verwendet werden, abgeleitet. Die Kategorien wurden dann weiter zusammengefasst, um eine hierarchische Struktur mit verschiedenen Abstraktionsebenen zu erhalten.

Schließlich wurden den Knoten Kohärenzwerte zugewiesen. Zunächst wurden entsprechend den Ausführungen in Abschnitt 4.1.4 Werte zwischen 0 und 1 für die Kategorien jedes Aspektes vergeben. Dann wurden den Sub-Graphen entsprechend der Wichtigkeit der jeweiligen Aspekte Gewichte zugewiesen. Der Aspekt „Semantik“ erhielt den Wert 1,0, um dessen größere Wichtigkeit gegenüber dem Aspekt „Visuelle Darstellung“ mit einem Wert von 0,4 herauszustellen. Der später für die Suche verwendete Kohärenzwert für jede Kategorie wurde dann durch Multiplikation des vorläufigen Kohärenzwertes mit dem Wert für die Wichtigkeit des jeweiligen Sub-Graphen angepasst. Dadurch kann später die relative Wichtigkeit noch modifiziert werden ohne die Werte für alle Kategorien ändern zu müssen.

Die Konstruktion eines initialen ERG nahm weniger als einen Tag in Anspruch. Für die Implementierung wurde er später ins RDF-Format (siehe Abschnitt 3.1.3) umgewandelt. Im Laufe der Entwicklung wurden zusätzlich diverse Anpassungen vorgenommen.

Im Folgenden beschäftigen wir uns zunächst mit Apache Lucene und der Indexierung, um das nötige Wissen über das von Lucene verwendete Retrievalmodell und die Indexstrukturen zu vermitteln. Bei der dann folgenden Beschreibung in Abschnitt 4.4 zeigen wir, wie Element Relationships letztendlich bei der Suche eingesetzt werden.

4.2 Apache Lucene

Apache Lucene [ApLu05] ist eine leistungsfähige Bibliothek für Suchmaschinen in Java, die für vielfältige Zwecke verwendet werden kann. Sie steht unter der Apache Software License zur Verfügung. Dabei ist Lucene keine sofort einsetzbare Anwendung, sondern stellt Klassen zur Verfügung, mit der Suchmaschinen für eine Vielzahl von Anwendungen implementiert werden können. Die API von Lucene kann dafür zur Indexierung und Suche beliebiger Dokumente verwendet werden.

Lucene wurde zwischen 1997 und 1998 von Doug Cutting geschrieben und ist heute Teil des Apache-Projektes, das sich als Sammelpunkt für diverse Open-Source-Lösungen in Java versteht. Die Entwicklung von Lucene wird durch freiwillige Mitarbeiter bei Apache vorangetrieben. Für die Implementierung verwenden wir die aktuelle Version 1.4.

4.2.1 Indexierung mit Lucene

Lucene speichert im Index eine Menge von Dokumenten. Die Dokumente werden innerhalb des Index über eine Dokumentnummer identifiziert, die von Lucene vergeben wird. Ein Dokument im Sinne von Lucene besteht aus einer Menge sogenannter Felder. Ein Feld ist dabei eine Sequenz von Termen, die durch einen Namen identifiziert wird. Die Terme werden von Lucene in Form einer invertierten Liste (siehe Abschnitt 2.4.1) gespeichert.

Für einzelne Felder kann bestimmt werden, ob diese Felder mit in den Index aufgenommen werden sollen (*indexed*) oder ob der Inhalt des Feldes nicht-invertiert im Index gespeichert werden soll (*stored*). Beliebige Kombinationen von *indexed* und *stored* sind dabei möglich. Zusätzlich kann bestimmt werden, ob der Text des Feldes in einzelne Token zerlegt werden soll (*tokenized*) oder in seiner Gesamtheit in den Index aufgenommen werden soll. In den meisten Fällen wird man den Text zerlegt haben wollen, in Einzelfällen, zum Beispiel zur Speicherung von Datumsangaben oder Dateinamen, kann es jedoch sinnvoll sein, das Feld nicht zu zerlegen.

Der Index der durch Lucene erzeugt wird, kann aus verschiedenen Sub-Indizes bestehen, die hier Segmente genannt werden. Jedes Segment speichert

- die Namen der Felder, die in den Dokumenten benutzt werden,
- die Werte derjenigen Felder, die als *stored* markiert wurden, als Attribut-Wert-Paar zugänglich durch die Dokumentnummer,
- ein Term-Lexikon, das alle Terme der indexierten Felder der Dokumente enthält, zusammen mit der Anzahl der Dokumente, die den Term enthalten (*document frequency*), wobei jeder Term Zeiger auf die Vorkommenshäufigkeits- und Positionsinformationen besitzt,
- die Vorkommenshäufigkeit der Terme, das heißt für jeden Term des Lexikons die Nummern der Dokumente, in denen der Term auftritt zusammen mit der Anzahl des Auftretens innerhalb dieser Dokumente (*term frequency*),
- Positionsinformationen der Terme, also für jeden Term des Lexikons die Position an denen er innerhalb der Dokumente auftritt und
- einen Normalisierungsfaktor (*boost*), mit dem für jedes Feld ein Wert bestimmt werden kann, mit dem der Relevanzwert für einen Treffer in diesem Feld multipliziert wird.

Bei der Indexierung muss immer eine Instanz der *Analyzer-Superklasse* angegeben werden, die dann bestimmt, wie der Text von Feldern, die als *tokenized* markiert sind, verarbeitet wird. Das bedeutet, Schritte wie Stammformreduktion und Stoppworteliminierung lassen sich durch eine konkrete Ausprägung dieser Klasse festlegen.

4.2.2 Suche mit Lucene

In diesem Abschnitt werden wir die zwei für uns bei der Suche wichtigen Punkte Anfragesprache und Retrievalmodell behandeln.

Lucene bietet durch den *QueryParser* bereits eine recht weit entwickelte Syntax für eine Anfragesprache. Die Suche nach einzelnen Termen wird ebenso unterstützt wie die Phrasensuche. Zusätzlich lassen sich boolesche Operatoren in der Anfrage verwenden. Die Suche kann auch auf einzelne Felder eingeschränkt werden. Details finden sich auf [ApLu05].

Lucene implementiert eine vereinfachte Form des Vektorraummodells bei der Suche mit einer Bewertung der Ergebnisse nach Relevanz. Dazu werden intern die Funktionen einer abstrakten *Similarity-Subklasse* aufgerufen, derart dass sich folgende Formel ergibt:

$$\text{sim}(q, d) = \sum_{t \in q} tf(\text{t in d}) \cdot idf(\text{t}) \cdot lengthNorm(\text{t.field in d}) \cdot tf(\text{t in q}) \cdot idf(\text{t}) \cdot queryNorm(\text{q}) \cdot coord(\text{q, d}) \cdot getBoost(\text{t.field in d})$$

Die Funktionen der Similarity-Klasse können durch Subklassen überschrieben werden, so dass sich die Berechnung der Ergebnisse in gewissen Grenzen beeinflussen lässt. In der Standardimplementierung der Similarity-Klasse, *DefaultSimilarity* werden die in der Formel verwendeten Funktionen wie folgt berechnet:

1. $tf(f_d) = \sqrt{f_d}$
2. $idf(f_D, N) = \ln\left(\frac{N}{f_D + 1}\right) + 1$
3. $lengthNorm(name_f, n_t) = \frac{1}{\sqrt{n_t}}$
4. $tf(f_q) = 1$
5. $queryNorm(\sum_{t \in q} w_t) = \frac{1}{\sqrt{\sum_{t \in q} w_t}}$
6. $coord(o, o_{max}) = \frac{o}{o_{max}}$

Wobei

f_d	Vorkommenshäufigkeit eines Terms innerhalb eines Dokumentes
f_D	Anzahl der Dokumente, in denen ein Term auftritt
N	Gesamtanzahl der Dokument in der Kollektion
$name_f$	Der Name des Feldes
n_t	Anzahl der Terme innerhalb diese Feldes
f_q	Vorkommenshäufigkeit eines Terms innerhalb einer Anfrage
w_t	Gewichtung tf · idf eines Terms t aus einer Anfrage q
o	Anzahl der im Dokument gefundenen Terme aus einer Anfrage
o_{max}	Gesamtanzahl der Terme aus einer Anfrage

Die Funktion *getBoost()* liefert den Wert zurück, der dem Feld bei der Indexierung zugewiesen wurde.

Die Berechnung von sim(q,d) erreicht dadurch sehr gute Ergebnisse, die allerdings mit einigen Nachteilen verbunden sind. Wie man in der obigen Formel erkennen kann, ist die *lengthNorm* nur eine starke Vereinfachung gegenüber der Formel aus Abschnitt 2.5.2, da hier nur die Anzahl der Terme, nicht aber deren Gewichtung berücksichtigt ist. Zur Verdeutlichung treffen wir folgende vereinfachende Annahmen: Wir setzen $tf \cdot idf = w$ und lassen die Normalisierungsfunktionen *coord()* und *getBoost()* zunächst außer Acht. Ausserdem nehmen wir an, dass immer im gleichen Feld gesucht wird. Dann ergibt sich für die Berechnung des Relevanzwertes eines Dokumentes d_j bezüglich einer Anfrage q folgende Formel (vergleiche Abschnitt 2.5.2), wobei n_j die Anzahl der Terme innerhalb eines Dokumentes ist:

$$\text{sim}(d_j, q) = \frac{\sum_{i=1}^{t} w_{i,j} \cdot w_{i,q}}{\sqrt{n_j} \cdot \sqrt{\sum_{j=1}^{t} w_{i,q}^2}}$$

Der damit verbundene Effekt ist, dass die Ergebnisse nicht mehr zwischen 0 und 1 liegen. Lucene normalisiert im Anschluss an eine Suche die Relevanzwerte zwar wieder auf den Bereich zwischen 0 und 1, allerdings lassen sich dadurch die Ergebnisse verschiedener Suchdurchläufe nicht

mehr miteinander vergleichen. Auf die Reihenfolge der Ergebnisse innerhalb eines Suchdurchlaufs hat dies aber keine Auswirkungen.

4.3 Indexierung

Bevor eine Suche durchgeführt werden kann, müssen geeignete Indexstrukturen aufgebaut werden, die es uns erlauben, XML-Dokumente mit Lucene zu durchsuchen und es uns ermöglichen, die im vorangegangenen Kapitel vorgestellten Operationen durchzuführen. Dabei kommt es im Einzelnen auf folgende Punkte an:

- Jedes Fragment des Dokumentes muss in Form einer invertierten Liste im Index abgelegt werden.
- Zu jedem Fragment müssen im Index Informationen über den Dateinamen des Dokumentes, aus dem das Fragment stammt, sowie dessen Position im Dokument in Form des XPath oder ähnlichen Informationen gespeichert werden.
- Insbesondere müssen die Informationen über Inline-Elemente verfügbar bleiben, um die Element Relationships berechnen zu können.

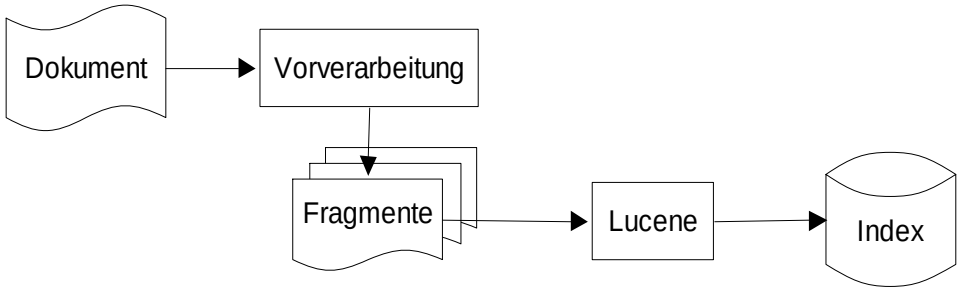
Da uns als Grundlage für die Indexierung nur die von Lucene unterstützten Mechanismen zur Verfügung stehen, musste ein Weg gefunden werden, diese entweder so anzupassen oder geeignet zu verwenden, dass die Indexierung von XML-Dokumenten möglich wurde.

Versuche, die interne Struktur von Lucene mit den für das XML-Retrieval nötigen Informationen anzureichern, wurden fallengelassen. Gründe hierfür sind zunächst der dafür notwendige Zeitaufwand, aber auch das Bestreben, die recht gute Performance von Lucene nicht durch zu viel zusätzlichen Overhead zu verschlechtern.

4.3.1 Prinzip der Indexierung

Daher wurde ein anderer Weg gewählt, der die Funktionalität von Lucene weitestgehend unverändert übernimmt. Abbildung 10 verdeutlicht die Architektur der Indexierung. Die Eingabedokumente werden durch eine Vorverarbeitung in geeignete Fragmente zerlegt, die dann an Lucene übergeben werden. Lucene übernimmt dann die Speicherung im Index. Der wichtigste Aspekt dabei ist die Vorverarbeitung, die es uns ermöglicht, geeignete Indexstrukturen aufzubauen.

Abbildung 10 Architekturüberblick Indexierung



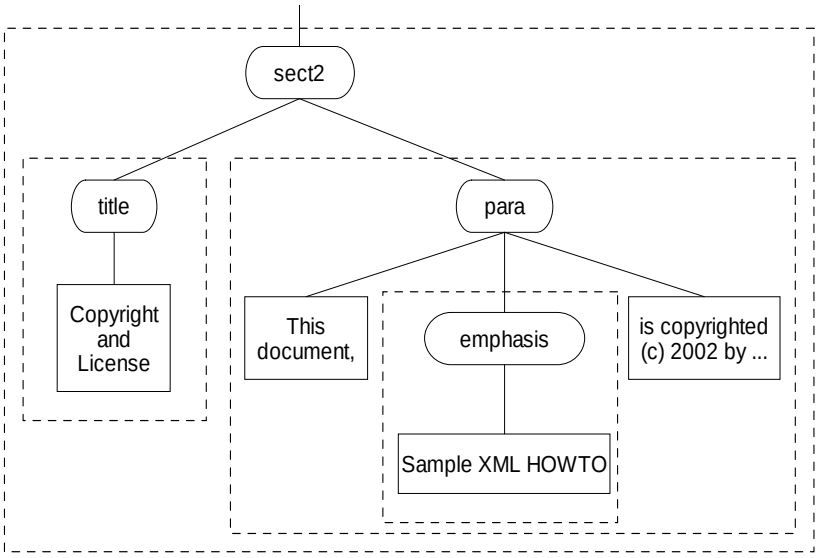
Um den genauen Ablauf der Indexierung zu verdeutlichen, betrachten wir das Dokumentfragment aus Beispiel 12.

Beispiel 12 XML-Dokumentfragment

```
<sect2 id="copyright">
<title>Copyright and License</title>
<para>
This document, <emphasis>Sample XML HOWTO</emphasis>,
is copyrighted (c) 2002 by ...
</para>
</sect2>
```

Das Beispiel besteht aus einem Abschnitt, der einen Titel und einen Absatz enthält. Innerhalb des Absatzes ist ein Inline-Element („*emphasis*“) definiert. Aus diesem Fragment werden jetzt alle Fragmente indexiert, die einzeln über einen XPath ansprechbar sind. Das bedeutet in unserem Fall, dass sich die Fragmente sect2[1], sect2[1]/title[1], sect2[1]/para[1] und sect2[1]/para[1]/emphasis[1] isolieren lassen. Daraus ergeben sich die zu indexierenden Blöcke wie in Abbildung 11.

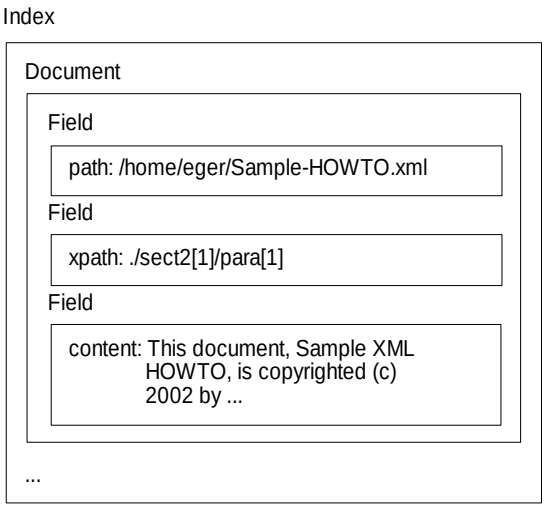
Abbildung 11 Baumdarstellung des Fragments aus Beispiel 12 mit zu indexierenden Blöcken



Im Index abgelegt wird nun für jedes Fragment nicht der komplette Inhalt mit den enthaltenen Tags, sondern nur enthaltene Text (*character data*) gespeichert, und zwar auch derjenige Text, der in den Subelementen enthalten ist. Attribute werden bei unserem Ansatz nicht berücksichtigt. Zunächst wird jedoch der Text in Kleinbuchstaben umgewandelt, eine Stoppwortelimination und eine Stammformreduktion nach Porter [Port80] durchgeführt. Für die Stoppwortelimination wurde eine erweiterte Stoppwortliste verwendet. Diese enthält erheblich mehr Wörter, als die von Lucene standardmäßig verwendete. Allerdings konnte dadurch für unsere Testkollektion nur eine Verkleinerung des Index um 7% erreicht werden, was noch einmal die Gültigkeit des Zipf'schen Gesetzes (siehe Abschnitt 2.5) verdeutlicht.

Der Index enthält damit Dokumente mit mehreren Feldern wie in Abbildung 12. In einem Feld wird der Pfad und Dateiname des Ursprungsdokumentes gespeichert. Ein weiteres Feld enthält den XPath zum aktuellen Fragment und ein drittes Feld enthält schließlich den Inhalt. Pfad und XPath werden nicht indexiert und dienen nur als Metainformationen zur Identifikation des betreffenden Fragmentes. Der Inhalt wird nur indexiert abgelegt (*unstored*), um den Index nicht unnötig aufzublähen. Man kann den Inhalt aus den Informationen über Dateiname und XPath zusammen mit den Ursprungsdokumenten später wieder rekonstruieren.

Abbildung 12 Speicherung des Fragmentes /sect2[1]/para[1] im Index

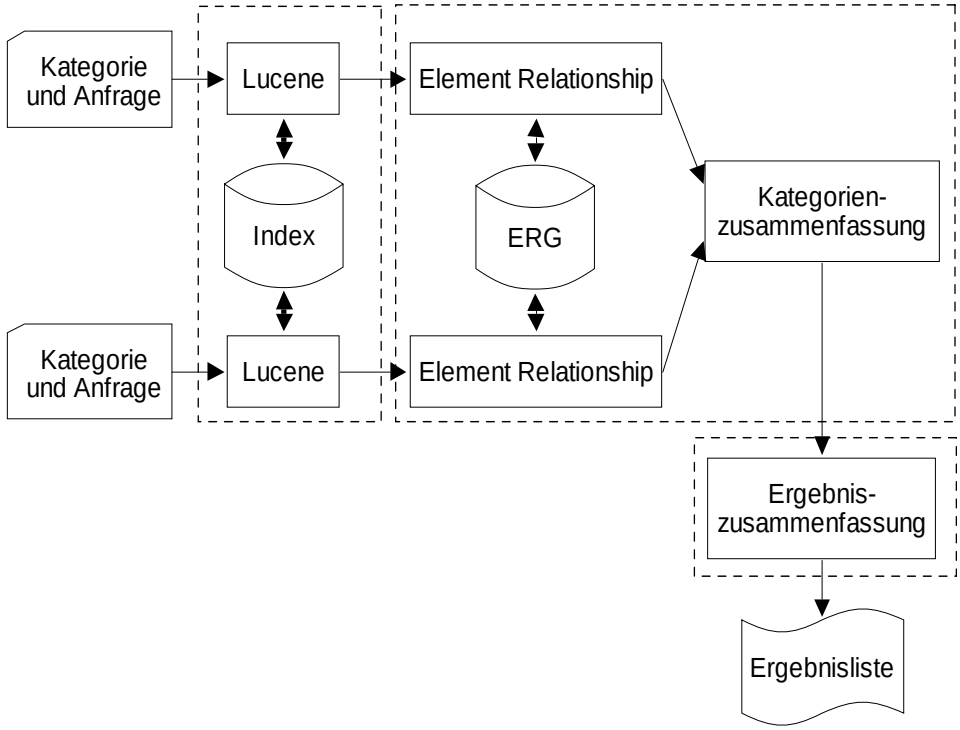


Wie in Abschnitt 4.2.1 bereits angesprochen berechnet Lucene bei der Indexierung bereits die Dokumenthäufigkeit (DF) und die Termhäufigkeit (TF). Da wir jedoch keine vollständigen Dokumente, sondern Dokumentfragmente indexieren, bezieht sich die Termhäufigkeit auf das Auftreten eines Termes nun nicht innerhalb eines Dokumentes, sondern innerhalb eines Fragmentes. Für die Dokumenthäufigkeit ergibt sich durch die von uns verwendete Art der Indexierung die Zahl der indexierten Fragmente, in denen der Term auftritt. Damit wird erreicht, dass für einen Term t im Index die von Mass und Mandelbrod [MaMa03] verwendete und in Abschnitt 3.3.1 bereits angesprochene Fragmenthäufigkeit CF(t) und Termhäufigkeit $TF_C(t)$ gespeichert wird.

4.4 Suche

Die Suche läuft in mehreren voneinander unabhängigen Schritten wie in Abbildung 4.5 ab. Die drei Blöcke kennzeichnen die Komponenten der Implementierung. Als Eingabe dienen beliebig viele Suchanfragen mit den dazugehörigen Kategorien. Anfragen ohne Angabe von Kategorien sind ebenfalls zulässig.

Abbildung 13 Architekturüberblick Suche



Diese Anfragen werden nach gleichen Kategorien zusammengefasst und dann einzeln an Lucene übergeben. Lucene führt nun für jede diese Anfragen eine Suche auf dem Index durch. Darin besteht ein wesentlicher Unterschied zum Konzept aus [Dopi05]. Dort wurde vorgeschlagen, die Suche nur auf den Elementen der aktuell betrachteten Kategorie durchzuführen. Diese Idee hat sich allerdings als nicht praktikabel herausgestellt, da gerade das Einlesen der Ergebnisse aus dem Index einen Geschwindigkeitsengpass darstellt (siehe Abschnitt 6.2.2).

Die Ergebnisse dieser einfachen Suche bestehen aus Dokumentfragmenten d_j. Diese Fragmente werden dann zunächst einzeln nach Kategorien an die Komponente übergeben, die die Berechnung der Element Relationships durchführt. Dabei werden folgende Schritte ausgeführt:

- Zunächst werden für jede Kategorie der Anfrage die Markup-Elemente bestimmt, die zu dieser Kategorie dazugehören. Dann wird der Relevanzwert der Ergebnisse, die in eines dieser Elemente eingebettet sind, mit dem entsprechenden Kohärenzwert multipliziert. Sofern es möglich ist, dass durch ein Erweitern der Kategorien noch Dokumente mit einem höheren Relevanzwert gefunden werden, wird dieser Schritt mit der erweiterten Kategorie nochmals durchgeführt. Damit ergibt sich:

$$sim_c(d_j, q) = sim(d_j, q) \cdot coherence(d_j, c)$$

  Alle übrigen Ergebnisse, die nicht in einer der Kategorien gefunden wurden, werden mit einem Wert $coherence_{min}$ abgewertet. Dies entspricht einer impliziten Kategorie auf oberster Ebene, die alle anderen Kategorien vereint und einen Kohärenzwert von $coherence_{min}$ hat.

- Dann werden die Ergebnisse der einzelnen Kategorien c zu einem Gesamtergebnis zusammengefasst. Dabei werden in der Anfrage optional übergebene Gewichtungen der Kategorien w_c berücksichtigt. Die Formel für diese Zusammenfassung lautet:

$$\sum_{c \in C} \frac{sim_c(d_j, q) \cdot w_c}{\sum_{c \in C} w_c} \cdot \left| \{ (c \in C) | sim_c \neq 0 \} \right|$$

  In einem letzten Schritt werden die Ergebnisse mittels einiger Heuristiken von Überlappungen bereinigt und die Ergebnisse werden zurückgeliefert. Eine ausführliche Beschreibung befindet sich im folgenden Abschnitt.

4.5 Zusammenfassen der Ergebnisse

Ein Zusammenfassen der Ergebnisse wird aus zwei Gründen erforderlich: Zum einen um nicht zu viele überlappende Ergebnisse zurückzuliefern. Diese Thematik wurde bereits in Abschnitt 3.3.2 diskutiert. Zum anderen liefert die Suche viele Ergebnisse zurück, die nur aus einem Inline-Element bestehen. Beispiel 13 veranschaulicht diese Situation.

Beispiel 13 **Häufiges Auftreten von Inline-Elementen in den Ergebnissen**

Wird eine Suchanfrage „*bash*“ (wahlweise mit oder ohne Kategorie) gestellt, so gibt es sehr viele Ergebnisse, die nur den Suchbegriff in ein Inline-Element eingeschlossen enthalten. Ein solches Ergebnis könnte beispielsweise sein:

```
<application moreinfo="none">bash</application>
```

Im Sinne des Vektorraummodells stellt dieses Ergebnis die Optimalsituation dar: Die Anfrage und das Ergebnis bilden im Vektorraum einen Winkel von 0° und damit einen Cosinus, der ja vereinfacht gesagt den Relevanzwert darstellt, von 1. Aus Benutzersicht ist dieses Ergebnis jedoch offensichtlich katastrophal.

Im traditionellen Information Retrieval stellt dies kein Problem dar, da dort die Dokumente der Kollektion und damit die Dokumente im Index von vornherein groß verglichen mit den Anfragen sind. Ein Dokument, das nur ein oder mehrere Wörter enthält, sollte dort nicht auftreten.

Daher verwenden wir verschiedene Heuristiken, um die Ergebnisse nachträglich zu verbessern. Wichtig ist hierbei, dass wir nur Ergebnisse entfernen, aber keine weiteren Veränderungen am Relevanzwert vornehmen.

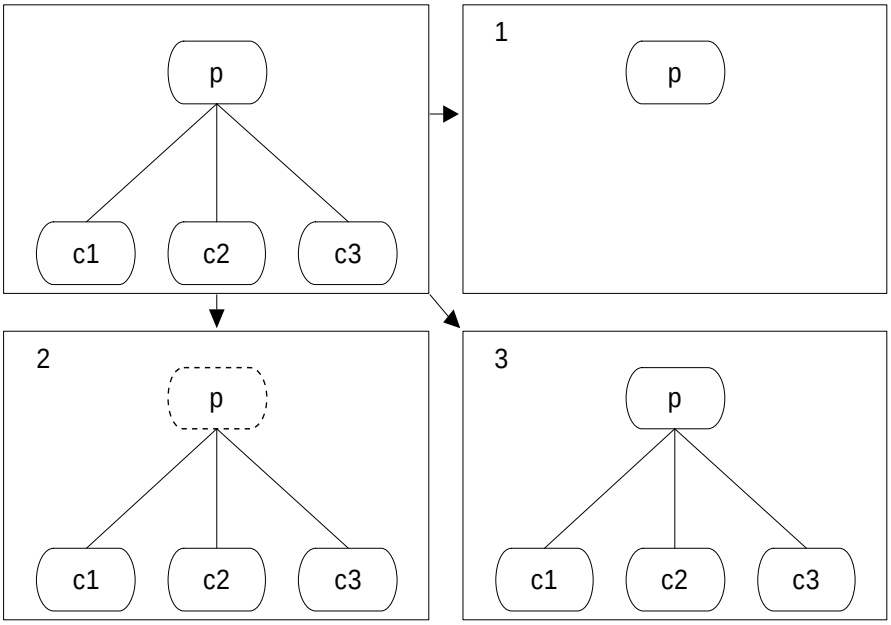
Es wird immer ein Knoten zusammen mit seinen direkten Nachkommen betrachtet. Wir bearbeiten die tiefsten Ergebnisse zuerst, das heißt die Ergebnisse mit dem längsten XPath. Dabei kann es zu drei verschiedenen Entscheidungen kommen (siehe Abbildung 14):

1. Zusammenfassen nach oben: Die Kindknoten werden aus dem Ergebnis entfernt; der Vaterknoten bleibt in der Ergebnismenge.

2. Zusammenfassen nach unten: Der Vaterknoten wird als „nicht im Ergebnis“ markiert und in einem späteren Schritt aus der Ergebnismenge entfernt; nur die Kindknoten bleiben in der Ergebnismenge.

3. Keine Zusammenfassung: Alle Knoten bleiben in der Ergebnismenge.

Abbildung 14 Verschiedene Fälle der Ergebniszusammenfassung



Wir werden im Folgenden die Bedingungen für die jeweiligen Entscheidungen einzeln näher erläutern. Zunächst erläutern wir jedoch die gesonderte Behandlung von Inline-Elementen bei der Ergebniszusammenfassung.

4.5.1 Behandlung von Inline-Elementen

Wir gehen in unserem Ansatz davon aus, dass Inline-Elemente nicht als gültiges Ergebnisgranulat zulässig sind. Daher werden diese bei der Ergebniszusammenfassung anders behandelt als andere Elemente. Treten in den Kindknoten eines Knotens, über dessen Zusammenfassung entschieden wird, Inline-Elemente auf, so werden diese bei der folgenden Betrachtung der Bedingungen zur Zusammenfassung nicht berücksichtigt. Die textuelle Länge dieser Inline-Elemente wird von der des Vaterknotens subtrahiert, um eine sinnvolle Berechnung der Überdeckung der übrigen Elemente zu ermöglichen. Dann wird mit den übrigen Kindknoten über eine Zusammenfassung entschieden.

Die Inline-Elemente werden als „nicht im Ergebnis“ markiert und dadurch später nicht zur Ergebnismenge hinzugefügt. Damit wir dadurch keine Ergebnisse verlieren, ist bei Knoten, deren Kindknoten Inline-Elemente enthalten, eine Zusammenfassung nach unten nicht erlaubt.

4.5.2 Zusammenfassen nach oben

Ein Zusammenfassen der Ergebnisse nach oben wird immer dann erforderlich, wenn alle Kindknoten in der Ergebnismenge vorhanden sind, beziehungsweise wenn sie einen genügend großen Teil des Vaterknotens überdecken. Betrachten wir dazu Beispiel 14, einen Abschnitt mit einem Titel und zwei Absätzen.

Beispiel 14 **Fast vollständige Überdeckung des Vaterknotens durch die Kindknoten (aus Sample-HOWTO.xml)**

```
<sect2 id="disclaimer">
  <title>Disclaimer</title>
  <para>
    No liability for the contents of this document can be accepted.
    Use the concepts, examples and information at your own risk.
    There may be errors and inaccuracies, that could be damaging to
    your system.  Proceed with caution, and although this is highly
    unlikely, the author(s) do not take any responsibility.
  </para>
  <para>
    All copyrights are held by their by their respective owners,
    unless specifically noted otherwise.  Use of a term in this
    document should not be regarded as affecting the validity of
    any trademark or service mark.  Naming of particular products
    or brands should not be seen as endorsements.
  </para>
</sect2>
```

Nehmen wir an, dass die beiden Absätze in der Ergebnismenge enthalten sind, der Titel jedoch nicht. Man sieht, dass eine genügend große Überdeckung des Vaterknotens gegeben ist, das heißt es erscheint hier sinnvoll, nur den Vaterknoten, also den ganzen Abschnitt in das Ergebnis zu übernehmen und nicht zusätzlich noch seine beiden Kindknoten.

In unserem Ansatz knüpfen wir jedoch noch eine weitere Bedingung an die Kindknoten. Angenommen, der erste Absatz erhält einen sehr hohen Relevanzwert, der zweite jedoch nur einen sehr geringen. In einem solchen Fall ist ein Zusammenfassen nicht sinnvoll, da dadurch ein gewisses Maß an Information verloren geht, nämlich darüber, dass der erste Absatz relevanter bezüglich der Anfrage ist als der zweite. Daher prüfen wir zusätzlich, ob die Relevanzwerte der Kindknoten nicht zu sehr voneinander abweichen.

Das bedeutet, eine Zusammenfassung nach oben und damit ein Streichen der Kindknoten aus der Ergebnismenge wird genau dann durchgeführt, wenn

- $\sum_{c \in C} \text{length}_c > \text{length}_p \cdot \text{thresh}_{\text{length}}$, wobei $\text{thresh}_{\text{length}}$ ein Wert zwischen 0 und 1 ist und
- $\sqrt{\frac{1}{|C|} \cdot \sum_{c \in C} (\text{sim}_{\text{mean}} - \text{sim}_c)^2} < \text{thresh}_{\text{sim}}$, wobei $\text{thresh}_{\text{sim}}$ ein Wert zwischen 0 und 1 ist.

4.5.3 Zusammenfassen nach unten

Trifft der erste Fall nicht zu, so wird überprüft, ob möglicherweise der Vaterknoten verworfen werden soll und nur die Kindknoten in der Ergebnismenge bleiben sollen. Im Prinzip versuchen wir zu diesem Zweck, aus den Relevanzwerten der Kindknoten den Relevanzwert des Vaterknotens zu schätzen. Gelingt uns dies oder übertrifft unsere Schätzung den Relevanzwert des Vaterknotens, so kann dieser nicht mehr Informationen enthalten als die einzelnen Kindknoten. Dann ist es sinnvoll ihn aus dem Ergebnis zu entfernen, da dieser Schritt nach dem Versuch der Zusammenfassung nach oben durchgeführt wird und somit klar ist, dass der Vaterknoten in diesem Zusammenhang viel irrelevante Information enthalten muss. Betrachten wir dazu 4.5.4.

Beispiel 15 **Relevanz der Kindknoten im Vergleich zum Vaterknoten (aus Upgrade.xml)**

```
<sect1>
  <title>Review security.</title>
  <para>
    (Sigh...) When I wrote this, this step was important but not
    crucial; the Internet was a friendlier place even in 1996 than
    it is today. Now, if your machine has Internet access, this
    step is utterly vital, and there are whole books devoted to
    getting it right; I can do nothing more here than offer a few
    very basic pointers:
  </para>
  <para>
    Check file permissions and directory permissions to be sure
    that access is neither too restricted nor too easy.  I find
    that Slackware tends to lean toward a more open environment
    than I like, so I go around changing 755's to 711's for
    binaries in the .../bin directories and stuff like that.  Or
    even 700's in the .../sbin ones.  Especial care is needed if
    you've carried over ftp, telnet or web servers; but then, if
    you were running those, you probably thought of that already.
  </para>
  <para>
    Look at /etc/inetd.conf or /etc/xinetd.conf and make sure
    you're not running any Internet services you don't need to.
    Also go through the boot scripts in /etc/rc.d and friends for
    the same purpose.  Check your firewall rules if your box is an
    Internet gateway or has Internet access.
  </para>
</sect1>
```

Wir nehmen an, dass zwei Suchdurchläufe durchgeführt wurden, einmal mit „*firewall rules*“ und einmal mit „*file permissions firewall rules*“.

Beim ersten Suchdurchlauf mit „*firewall rules*“ ist der gesuchte Begriff vollständig im dritten Absatz enthalten. Der Relevanzwert des Abschnittes berechnet sich daher nur aus dem Inhalt dieses Absatzes. In diesem Fall kann daher nach unten zusammengefasst werden, da durch den Abschnitt nicht mehr Information bezüglich der Anfrage vermittelt wird als durch den Absatz.

Der zweite Fall verdeutlicht die umgekehrte Situation. Eine Suchanfrage „*file permissions firewall rules*“ liefert sowohl im zweiten, als auch im dritten Absatz einen Treffer, allerdings jeweils nur teilweise. Der Abschnitt ist in diesem Fall das bessere Ergebnis, da hier durch den ganzen Abschnitt mehr Informationsbedarf aus der Anfrage befriedigt werden kann. Daher wird hier nicht nach unten zusammengefasst. Die einzelnen Absätze werden trotzdem im Ergebnis belassen, weil über eine Zusammenfassung nach oben im vorherigen Schritt bereits entschieden wurde.

Ein weiterer Faktor unserer Berechnung ist, dass Elemente, die durch die Berechnung der Element Relationships besser bewertet wurden eine größere Chance haben, nicht durch die Zusammenfassung nach unten mit ihren Kindknoten verschmolzen zu werden. Wenn Ergebnisse ein Inline-Element umschließen, das der in der Anfrage spezifizierten Kategorie angehört, so gehen wir davon aus, dass diese Ergebnisse eher als zusätzliche Ergebnisse erwünscht sind.

Ein Zusammenfassen nach unten wird genau dann durchgeführt, wenn

- die Bedingungen aus Abschnitt 4.5.2 nicht erfüllt sind und
- keine Inline-Elemente in den Kindknoten enthalten sind (siehe Abschnitt 4.5.1) und
- $\sum_{c \text{ in } C} (length_c \cdot sim_c) > length_p \cdot sim_p \cdot thresh_{length \cdot sim}$, wobei $thresh_{length \cdot sim}$ ein Wert zwischen 0 und 1 ist.

4.5.4 Keine Zusammenfassung

Konnte in den beiden vorangegangenen Schritten weder eine Entscheidung über das Zusammenfassen nach oben noch über das Zusammenfassen nach unten getroffen werden, findet keine Zusammenfassung des aktuellen Knoten und seinen direkten Nachkommen statt. Wir nehmen in diesem Fall an, dass sowohl für den Vater- als auch für die Kindknoten keine theoretische Grundlage für eine Zusammenfassung besteht und daher alle Ergebnisse eine gewisse Berechtigung haben, in der resultierenden Ergebnisliste aufgeführt zu werden.

Damit wird genau dann keine Zusammenfassung durchgeführt, wenn

- die Bedingungen aus Abschnitt 4.5.2 nicht erfüllt sind und
- die Bedingungen aus Abschnitt 4.5.3 nicht erfüllt sind.

4.5.5 Bestimmung der Schwellenwerte

Die Werte $thresh_{length}$, $thresh_{sim}$ und $thresh_{length \cdot sim}$ wurden empirisch durch Betrachtung überlappender Ergebnisse bestimmt. Wir verwenden

- $thresh_{length} = 0{,}98$,
- $thresh_{sim} = 0{,}3$ und
- $thresh_{length \cdot sim} = 0{,}6$.

Diese Werte liefern für die von uns verwendete Dokumentenkollektion gute Ergebnisse. Für andere Dokumentenkollektionen müssen sie aber möglicherweise anhand der Überlappung der Ergebnisse angepasst werden. Dabei ist zu beachten, dass stärker zusammengefasst wird für größere Werte von $thresh_{sim}$ und kleinere Werte von $thresh_{length}$ und $thresh_{length \cdot sim}$.

Kapitel 5 Implementierungsaspekte

Dieses Kapitel beschreibt die konkrete Implementierung der im vorangegangenen Kapitel vorgestellten Konzepte.

Die Implementierung besteht wie jede Suchmaschine aus den zwei Kernkomponenten Indexierung und Suche. Zunächst werden wir die für die Indexierung benötigten Klassen und deren Zusammenhang beschreiben, danach die für die Suche, aufgeteilt nach Berechnung der Element Relationships und Zusammenfassen der Ergebnisse. Abschließend wird die Bedienung der Benutzerschnittstellen für Indexierung und Suche erläutert.

5.1 Implementierungsdetails Indexierung

Die Indexierung wird entsprechend den Ausführungen aus Abschnitt 4.3 hauptsächlich von den Klassen des Pakets *index* durchgeführt. Zusätzlich existiert im Paket *demo* noch ein Kommandozeilenclient, der die Indexierung anstößt (siehe Abschnitt 5.3.1). Abbildung 15 zeigt die Zusammenarbeit der Klassen des Pakets *index* in einem Sequenzdiagramm. Prinzipiell werden die Eingabedokumente mittels SAX geparst und dem Indexierungsmechanismus von Lucene übergeben. Wir gehen nun im Einzelnen auf die beteiligten Klassen ein.

FileXMLDocument

Die Klasse *FileXMLDocument* besitzt nur den statischen Funktionsaufruf *addDocument()*, mit dem ein als Parameter übergebenes XML-Dokument durch den ebenfalls als Parameter übergebenen *XMLDocumentWriter* zum Index hinzugefügt werden kann.

Die Funktion instanziert zunächst einen SAX-Parser. SAX ist im eine ereignisbasierte API für das Parsen von XML-Dokumenten. Ein Dokument wird sequentiell von einem *XMLReader* eingelesen. Abhängig davon, welches syntaktische Konstrukt der *XMLReader* liest, werden bestimmte Callback-Methoden eines sogenannten *ContentHandler* aufgerufen. In unserem Fall ist der *ContentHandler* ein *XMLDocumentHandler.* Beispiel 16 soll die Funktionsweise von SAX verdeutlichen.

Dann startet die Methode das Parsing. Das Sequenzdiagramm aus Abbildung 15 beginnt mit dem Aufruf der Methode *parse()* des *XMLReader.*

Beispiel 16 Aufruf der Callback-Methoden durch SAX

Betrachten wir das folgende XML-Dokument:

```
<?xml version="1.0"?>
<article>
<para>SAX parsing example</para>
</article>
```

Beim Parsen dieses Dokumentes werden vom Parser die folgenden Methoden des DocumentHandler aufgerufen:

```
startDocument()
startElement("article")
startElement("para")
characters("SAX parsing example")
endElement("para")
endElement("article")
endDocument()
```

XPathStack

Der XPathStack ist eine Erweiterung der Klasse *java.util.Stack* und wird von der Klasse *XMLDocumentHandler* benötigt, um beim Parsen eines Dokumentes den korrekten XPath herausfinden zu können. Die Verarbeitung läuft folgendermaßen ab: Mittels *push()* wird ein Markup-Element an die Klasse übergeben. Mit der *toString()*-Methode kann nun der XPath dieses Elementes ausgelesen werden, bis das Element mit *pop()* wieder aus dem Stack entfernt wird. Die Klasse kümmert sich intern darum, dass bei der Ausgabe die korrekte Position (zum Beispiel *„/article**[1]**/sect1**[2]**/sect2**[4]**“*) des Elementes im Dokument berechnet wird.

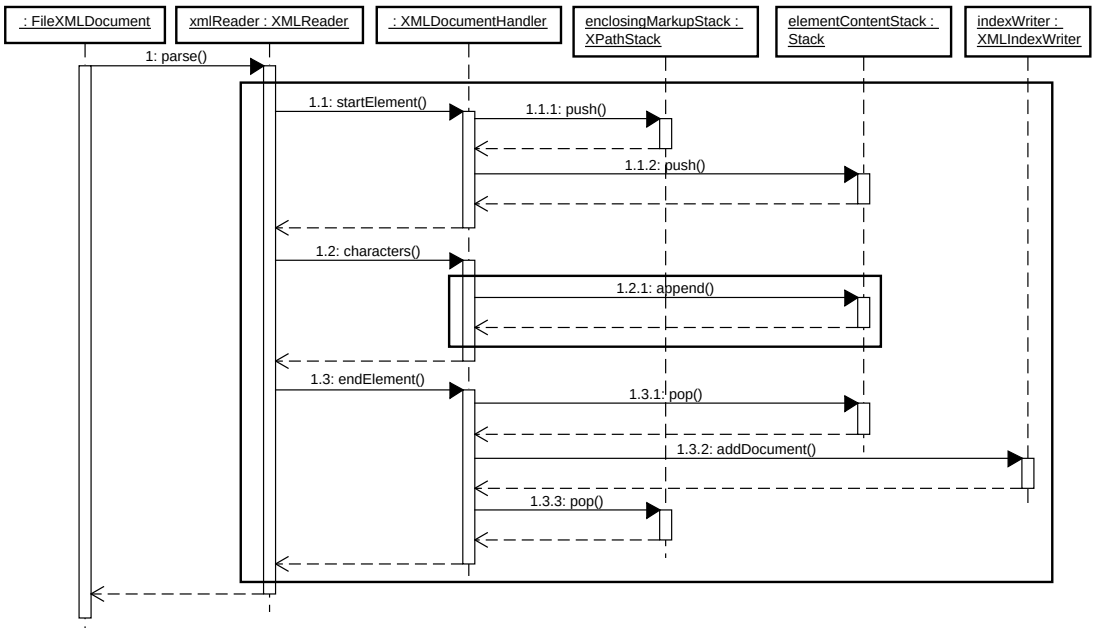
XMLDocumentHandler

Die Kernkomponente der Indexierung stellt die Klasse *XMLDocumentHandler* dar. Es handelt sich dabei um eine Implementierung des Interface *org.xml.sax.ContentHandler.*

Die Methoden, die dabei für uns von Relevanz sind, sind *startElement(), endElement()* und *characters()*. Die Klasse verwendet intern zwei Stacks, einen *XPathStack*, mit dem der XPath der Elemente verfolgt wird und einen *Stack (elementContentStack)*, mit dem der Inhalt der Elemente verfolgt wird.

Beispiel 18 verdeutlicht in einer vereinfachten Darstellung die Funktionsweise anhand des Dokumentfragmentes aus Beispiel 17. Abbildung 15 zeigt den Ablauf der Methodenaufrufe. Beim Aufruf des Ereignisses *startElement()* wird der Name des Elements auf den *XPathStack* gelegt und ein neuer *StringBuffer* auf den *elementContentStack*. Wird dann das Ereignis *characters()* ausgelöst, also das Auftreten von Text innerhalb dieses Elementes, so wird dieser Text an alle *StringBuffer*, die sich auf dem *elementContentStack* befinden, angehängt. Wird schließlich das Ereignis *endElement()* ausgelöst, wird das oberste Element vom *elementContentStack* entfernt und zusammen mit dem XPath, der sich ja durch die *toString()*-Methode des *XPathStack* ergibt, als Felder (*org.apache.lucene.document.Field*) zu einem Dokument (*org.apache.lucene.document.Document)* hinzugefügt, das der *XMLDocumentWriter* dem Index hinzufügt. Dann wird auch das oberste Element des *XPathStack* entfernt.

Abbildung 15 Sequenzdiagramm für die Indexierung



Außerdem werden noch folgende Felder in dem Dokument angelegt:

- Der Pfad und Dateiname des aktuellen Dokumentes.
- Das Datum der letzten Modifikation der Datei.
- Die Länge des Inhaltes in Zeichen.

XMLIndexWriter

Der *XMLIndexWriter* ist eine Subklasse von *org.apache.lucene.index.IndexWriter.* Diese Klasse besitzt eine Methode *addDocument()*, mit der ein vom *XMLDocumentHandler* vorbereitetes Dokument in den Index geschrieben werden kann. Die aktuelle Implementierung ist nur ein Stub, der eingeführt wurde, um eine spätere Erweiterbarkeit der Indexierung zu gewährleisten. Denkbare Modifikationen an dieser Stelle beinhalten das Schreiben von angepassten oder erweiterten Indexstrukturen.

Beispiel 17 XML Beispieldokument (modifizierter Ausschnitt aus Sample-HOWTO.xml)

```
<sect1>
  <title>Copyright and License</title>
  <para>
    This document, <emphasis>Sample XML HOWTO</emphasis>,
    is copyrighted (c) 2002 by ...
  </para>
</sect1>
```

Beispiel 18 Funktionsweise des XMLDocumentHandler

Funktionsaufruf	elementContentStack	XPathStack.toString()	Übergabe an Lucene
startElement("sect1")	[]	/sect1[1]	
startElement("title")	[] []	/sect1[1]/title[1]	
characters("Copyright and License")	["Copyright and License"] ["Copyright and License"]	/sect1[1]/title[1]	
endElement("title")	["Copyright and License"]	/sect1[1]	/sect1[1]/title[1]; "Copyright and License"
startElement("para")	[] ["Copyright and License"]	/sect1[1]/para[1]	
characters("This document, ")	["This document, "] ["Copyright and License This document, "]	/sect1[1]/para[1]	
startElement("emphasis")	[] ["This document, "] ["Copyright and License This document, "]	/sect1[1]/para[1]/emphasis[1]	
characters("Sample XML HOWTO")	["Sample XML HOWTO"] ["This document, Sample XML HOWTO"] ["Copyright and License This document, Sample XML HOWTO"]	/sect1[1]/para[1]/emphasis[1]	
endElement("emphasis")	["This document, Sample XML HOWTO"] ["Copyright and License This document, Sample XML HOWTO"]	/sect1[1]/para[1]	/sect1[1]/para[1]/emphasis[1]; "Sample XML HOWTO"
characters("is copyrightet (c) 2002 by ...")	["This document, Sample XML HOWTO is copyrightet (c) 2002 by ..."] ["Copyright and License This document, Sample XML HOWTO is copyrightet (c) 2002 by ..."]	/sect1[1]/para[1]	
endElement("para")	["Copyright and License This document, Sample XML HOWTO is copyrightet (c) 2002 by ..."]	/sect[1]	/sect1[1]/para[1]; "This document, Sample XML HOWTO is copyrightet (c) 2002 by ..."
endElement("sect1")			/sect[1]; "Copyright and License This document, Sample XML HOWTO is copyrightet (c) 2002 by ..."

5.2 Implemetierungsdetails Suche

Die für die Suche (siehe Abschnitt 4.4) verwendeten Klassen befinden sich im Paket *search* und *search.hit.* Dabei ist das Paket *search* für die Suchfunktionalität im Allgemeinen verantwortlich. Klassen im Paket *search.hit* dienen zur Verwaltung und für Operationen auf Einzelergebnissen. Zusätzlich gibt es im Paket *demo* einen Kommandozeilenclient für die Suche. Außerdem wurden einige JSP-Seiten erstellt, die einen Webclient implementieren.

5.2.1 Allgemeine Klassen

Die Klassen in diesem Abschnitt sind von zentraler Bedeutung für die Implementierung. Sie werden an unterschiedlichen Stellen immer wieder benutzt und daher hier im Vorfeld kurz beschrieben.

Hit, HitList

Die Klasse *Hit* repräsentiert ein einzelnes Ergebnis. Sie speichert die Werte der Felder eines Lucene-Ergebnisdokuments (Pfad, XPath und Länge des Inhalts) zusammen mit dessen Relevanzwert und Kategorie. Außerdem gibt es verschiedene Methoden für den einfachen Zugriff auf die Felder des Dokuments.

Mit der Methode *equals()* können zwei *Hits* auf Identität bezüglich ihres Ergebnisdokumentes verglichen werden. Zusätzlich bietet die Methode *getXMLFragment()* die Möglichkeit, mittels des Dateinamens und des XPath aus dem Dokument den zugehörigen Dokumentinhalt aus der Datei zu laden.

HitList ist die Erweiterung einer *java.util.ArrayList* zur Speicherung von *Hits*, die direkten Zugriff auf die Getter-Methoden der *Hits* gewährt.

HitComparator

Der *HitComparator* dient zum Vergleichen von *Hits.* Er wird aber auch zum Sortieren der Ergebnislisten benötigt. Ein *Hit* kann damit zum Einen über den Relevanzwert verglichen werden, zum Anderen über Dateiname und XPath.

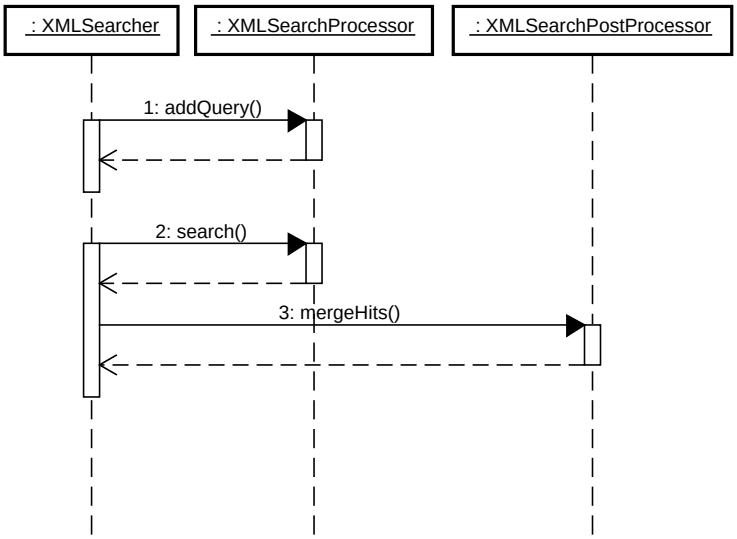
5.2.2 XMLSearcher

Die Klassen für den Zugriff auf die Suchfunktionalität werden von der Klasse *XMLSearcher* gekapselt. Diese Klasse bietet ein einfaches API zur Instanzierung einer Suche, zum Hinzufügen von Anfragen und zum Starten der Suche.

Nach Instanzierung der Klasse können mit der Methode *addQuery()* beliebig viele Anfragen mit den zugehörigen optionalen Kategorien zur Suche hinzugefügt werden. Diese Methode dient nur zur Kapselung und leitet den Methodenaufruf direkt an die Methode *addQuery()* der Klasse *XMLSearchProcessor* weiter (siehe Abbildung 16).

Ein Aufruf der Methode *search()* startet die Suche. Beim Aufruf der Methode ohne Parameter werden die Ergebnisse nach Relevanz sortiert, durch Übergabe eines geeigneten Parameter (*HitComparator.SORT_BY_XPATH*) können die Ergebnisse auch nach XPath sortiert werden. Als Rückgabewert wird eine *HitList* mit den Suchergebnissen geliefert. Die Methode ruft zunächst die Methode *search()* des *XMLSearchProcessor* auf, mit den von diesem gelieferten Ergebnissen wird dann die Methode *mergeHits()* des *XMLSearchPostProcessor* aufgerufen.

Abbildung 16 Sequenzdiagramm für die Suche



5.2.3 Berechnung der Element Relationships

XMLSearchProcessor

Der XMLSearchProcessor arbeitet die beiden Schritte „einfache Suche mit Lucene“ und „Berechnung der Element Similarity“ ab. Außerdem wird bei Instanzierung der Klasse der *ElementRelationshipGraph* initialisiert.

Durch die Methode *addQuery()* können Anfragen und deren Kategorien hinzugefügt werden. Für den Parameter der Kategorie wird eine optionale Gewichtung der Form *Kategoriename:Kategoriegewichtung* unterstützt, mit der man gewisse Kategorien bei der Suche bevorzugen kann.

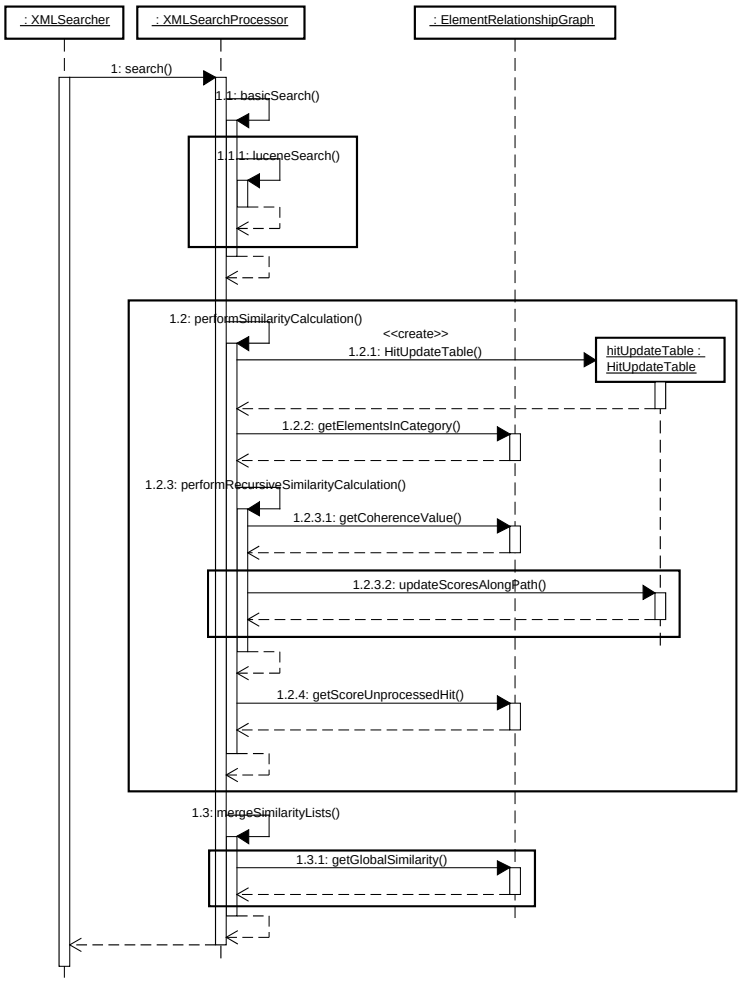
Die Methode *search()* (siehe Abbildung 17, zeigt den Ablauf ohne Kategorieerweiterung) ruft zunächst die Methode *basicSearch()* auf, die für alle Anfragen eine Suche mit Lucene durchführt (*luceneSearch()*). Die Ergebnisse werden in einer Hash-Tabelle mit den Kategorien als Schlüssel gespeichert. Dann wird für jede Kategorie einzeln die Methode *performSimilarityCalculation()* aufgerufen. Dort wird zunächst mit den Treffern der aktuellen Kategorie ein *HitUpdateTable* gefüllt. Dadurch können später für einen Treffer, der in einer Kategorie gefunden wurde, dessen Vaterelemente leicht mit dem Kohärenzwert aktualisiert werden. Danach werden aus dem ERG alle Markup-Elemente abgefragt, die in der angegebenen Kategorie liegen (*getElementsInCategory()*).

Die Anpassung der Relevanzwerte mit den Kohärenzwerten findet in der rekursiven Methode *performRecursiveSimilarityCalculation()* statt. Die Methode wird mit den Parametern Kategoriename, einer Liste der Elemente in der Kategorie und einer Liste von Ergebnissen aufgerufen. Zunächst wird über alle Ergebnisse iteriert, wobei jeweils überprüft wird, ob das aktuelle Ergebnis in eines der Markup-Elemente der Kategorie eingebettet ist. Wenn das der Fall ist, dann wird

der Relevanzwert des aktuellen Ergebnisses und allen seinen Vaterelementen entlang des XPath mittels *updateScoresAlongPath()* mit dem Kohärenzwert aus dem ERG multipliziert und die Ergebnisse als bearbeitet markiert. Sind nach einmaligem Durchlauf noch nicht alle Elemente als bearbeitet markiert, wird aus der *ElementRelationshipGraph*-Klasse abgefragt, ob eine Kategorieerweiterung sinnvoll ist (*broadeningCategoriesMakesSense()*, liefert in der aktuellen Implementierung immer *true* zurück) und gegebenenfalls die Methode erneut aufgerufen mit der erweiterten Kategorie und deren Elementen als neuem Parameter.

Nach Abarbeitung dieser Rekursion werden allen Ergebnissen, deren Relevanzwert noch nicht verändert wurde, mittels der Methode *getScoreUnprocessedHit()* der Klasse *ElementRelationshipGraph* ein neuer Relevanzwert zugewiesen, um diese Ergebnisse abzuwerten.

Abbildung 17 Sequenzdiagramm für die Lucene-Suche und Berechnung der Element Similarity



Die Methode *mergeSimilarityLists()* fasst schließlich die Ergebnisse verschiedener Kategorien zusammen. Die Relevanzwerte der einzelnen Ergebnisse werden durch die Methode *getGlobalSimilarity()* des ERG, wie in Abschnitt 4.4 angegeben, zusammengefasst.

ElementRelationshipGraph

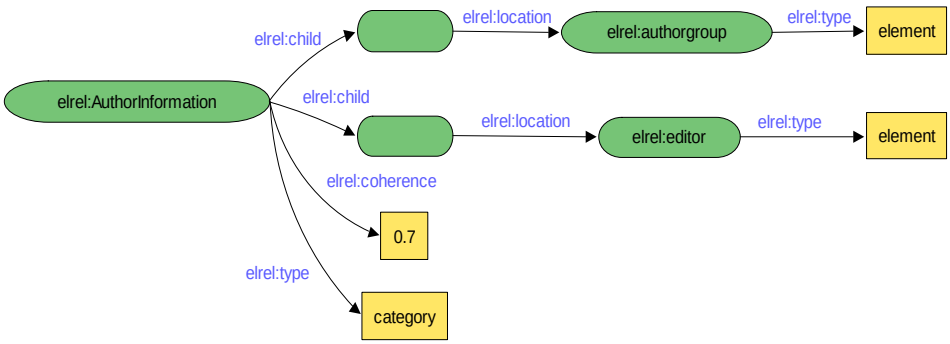
Die Klasse *ElementRelationshipGraph* (siehe Abschnitt 4.1) bietet Methoden zu Einlesen und Abfragen des ERG. Bei der Initialisierung wird der ERG einmal aus ein oder mehreren Dateien im RDF-Format eingelesen. Abbildung 18 zeigt einen Ausschnitt aus dem RDF-Graph. Die Kategorien, in der Abbildung *AuthorInformation*, können selbst wieder „*child*“ anderer Kategorien sein.

Als Subjekte sind Kategorien und Elemente erlaubt. Diese müssen ein Prädikat „*type*“ mit dem Wert „*category*“ oder „*element*“ besitzen. Subjekte mit dem Typ „*category*“ müssen ein weiteres Prädikat „*coherence*“ haben, dessen Wert dem Kohärenzwert der Kategorie entspricht. Kategorien können außerdem beliebig viele Prädikate „*child*“ haben, deren Objekt ein leerer Knoten ist. Dieser leere Knoten muss dann wieder ein Prädikat „*location*“ haben, dessen Objekt ein Element oder eine Kategorie ist.

In den verschiedenen Methoden zum Abfragen des ERG wird dann mittels RDQL, einer Anfragesprache für RDF, auf den Graph zugegriffen. Dabei wurden Mechanismen zur Zwischenspeicherung der Ergebnisse implementiert, da die RDQL-Anfragen nicht besonders effizient sind.

Die verschiedenen Methoden wie zum Beispiel *getElementsInCategory()* oder *getCoherenceValue()* sind nach konkreten Anforderungen aus der Implementierung erstellt worden. Das eigentliche Modell und damit die wichtigen Informationen befinden sich jedoch nur im RDF-Graph. Um für spätere Erweiterungen flexibel zu sein, bietet die Methode *queryERG()* eine Möglichkeit, auch ohne Änderungen am Quellcode der Klasse *ElementRelationshipGraph*, Anfragen an das Modell zu stellen.

Abbildung 18 Ausschnitt des RDF-Graph des ERG



HitUpdateTable

Der *HitUpdateTable* wird mit einer Liste von *Hits* initialisiert und wird dann dazu verwendet, den Relevanzwert für ein Ergebnis mit der Methode *updateScoresAlongPath()* entlang dessen XPath bis zum Wurzelknoten mit einem Wert zu multiplizieren. Diese Klasse wird vom *XMLSearchProcessor* dazu verwendet, den Relevanzwert aller Vorfahrenknoten eines Ergebnisses mit dem Kohärenzwert anzupassen.

5.2.4 Details des Zusammenfassens der Ergebnisse

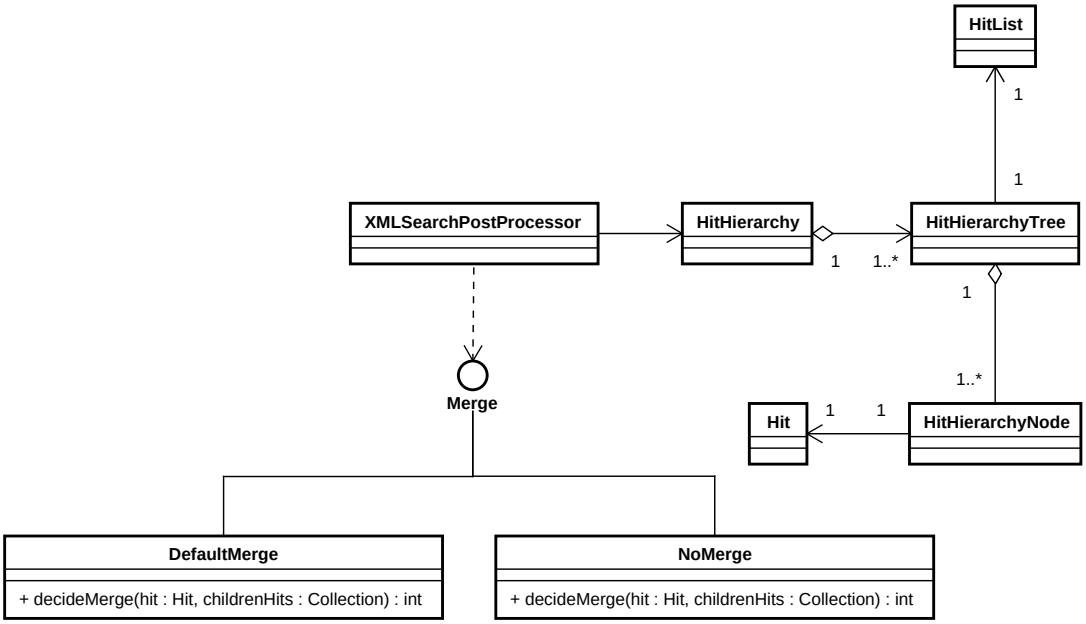
Nachdem der *XMLSearchProcessor* die Element Relationships bewertet hat, werden die Ergebnisse an den *XMLSearchPostProcessor* übergeben, der das Zusammenfassen der Ergebnisse steuert. Bevor wir auf die Details der Verarbeitung durch den *XMLSearchProcessor* eingehen, werden wir zunächst die Datenstruktur beschreiben, die wir zur Verwaltung der Ergebnisse benötigen.

HitHierarchy, HitHierarchyTree und HitHierarchyNode

Abbildung 19 verdeutlicht in einem Klassendiagramm den Zusammenhang dieser Klassen. In die *HitHierarchy* werden alle Ergebnisse eingefügt, um die zur Zusammenfassung der Ergebnisse nötigen Informationen bereitzuhalten. Die *HitHierarchy* besteht zunächst aus einer Hash-Tabelle. Dort werden als Schlüssel der Pfad und Dateiname der Ergebnisse und als Wert ein *HitHierarchyTree* für die Ergebnisse des betreffenden Dokuments eingefügt (siehe Abbildung 20). Die Knoten der Baumstruktur sind Instanzen der Klasse *HitHierarchyNode*.

Beim Einfügen eines *Hit* in den *HitHierarchyTree* wird an der korrekten Position im Baum eine *HitHierarchyNode* erzeugt, die den *Hit* kapselt. Die Struktur, die dabei aufgebaut wird, entspricht grob der in Abschnitt 3.1.1 angesprochenen Baumstruktur eines XML-Dokumentes. Sie unterscheidet sich dadurch, dass hier nur Elementknoten erzeugt werden und diese auch nur für solche Elemente, für die es in der Ergebnismenge ein *Hit*-Objekt gibt.

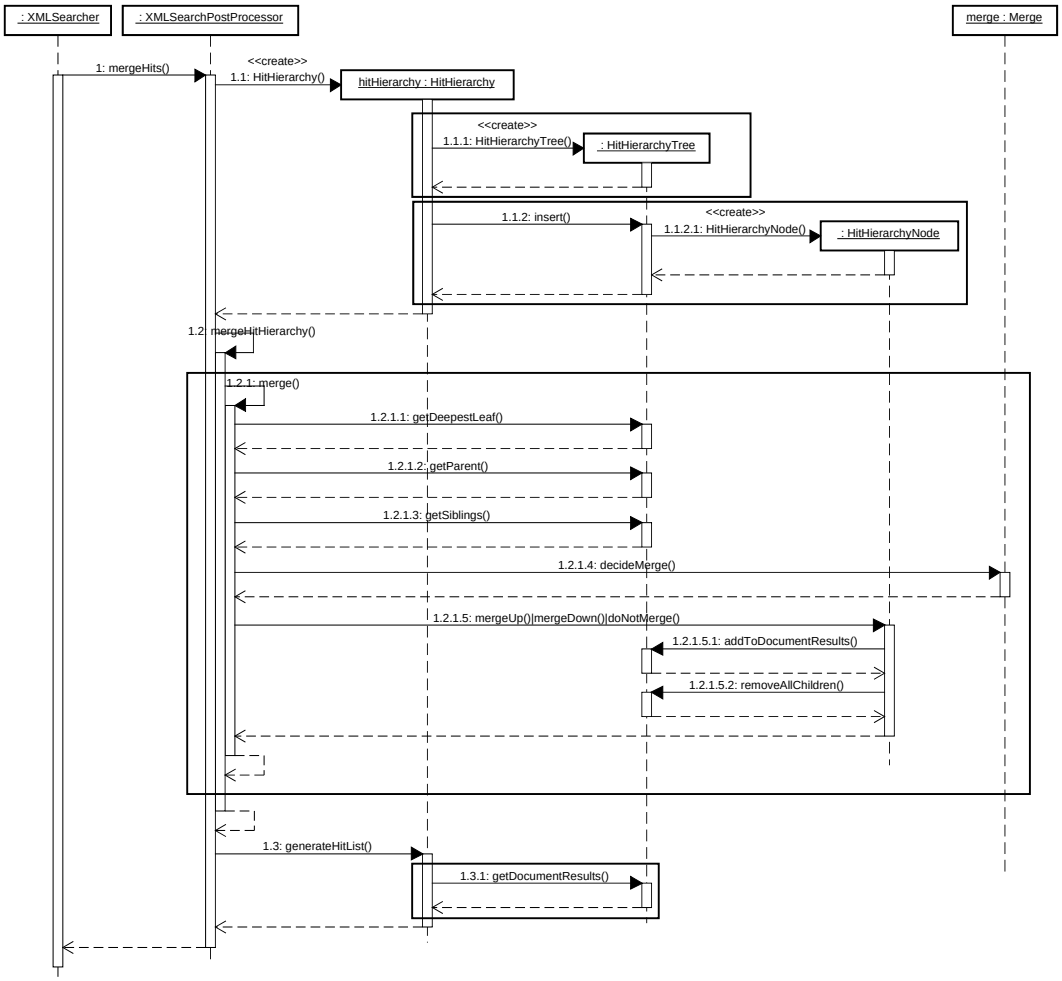
Abbildung 19 Klassendiagramm für das Zusammenfassen der Ergebnisse



Die Baumstruktur wird während des Zusammenfassens (siehe Abschnitt 4.5) durch die Methoden *mergeUp()*, *mergeDown()* und *doNotMerge()* der *HitHierarchyNodes* modifiziert. In jeder dieser Methoden werden die Kindknoten des angesprochenen Knotens aus dem *HitHierarchy-*

Tree entfernt. Dadurch wird während des Zusammenfassens der Baum immer weiter reduziert, bis kein Knoten mehr enthalten ist. Bei den Methoden *mergeDown()* und *doNotMerge()* werden zuvor mit der Methode *getChildrenHits()* die Hit-Objekte dieser Knoten gefunden und zu einer *HitList* des *HitHierarchyTree* hinzugefügt, die die späteren Ergebnisdokumente enhält. Die Methode *mergeDown()* kann zusätzlich die aktuelle *HitHierarchyNode* markieren, so dass sie bei einem späteren Aufruf der Methode *getChildrenHits()* auf der nächsthöheren Ebene im Baum nicht berücksichtigt werden.

Abbildung 20 Sequenzdiagramm für das Zusammenfassen der Ergebnisse



XMLSearchPostProcessor

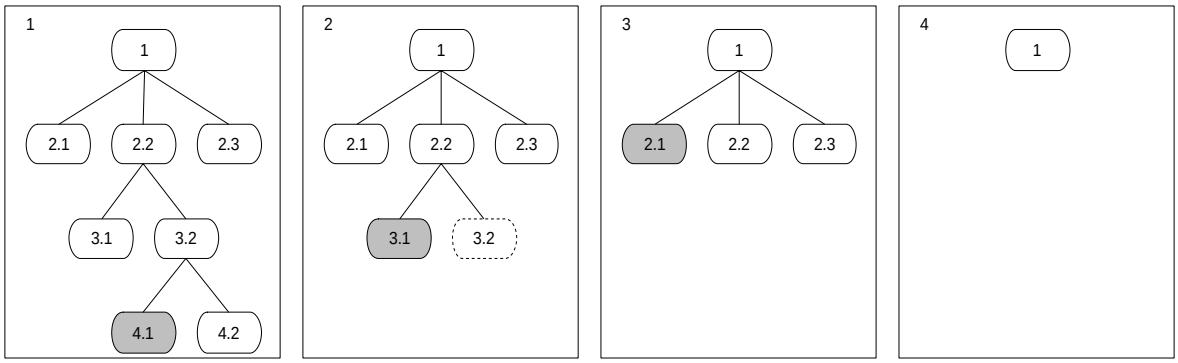
Der *XMLSearchPostProcessor* erzeugt zunächst eine *HitHierarchy* aus den im Funktionsaufruf übergebenen Ergebnissen (Abbildung 20). Für jedes Dokument, in dem es Ergebnisse gibt, werden nun die folgenden Schritte durchgeführt:

- Finde den tiefsten liegenden Blattknoten aus dem *HitHierarchyTree* dieses Dokumentes (*getDeepestLeaf()*).

- Übergebe den Vaterknoten und die Geschwisterknoten an die Methode *decideMerge()* einer Ausprägung des *Merge*-Interface.
- Rufe je nach Rückgabewert der Methode *decideMerge()* die entsprechenden Methoden *mergeUp()*, *mergeDown()* oder *doNotMerge()* des Vaterknoten des aktuellen Knotens auf.
- Wiederhole diese Schritte, so lange Knoten im *HitHierarchyTree* vorhanden sind.

Auf diese Art werden die *HitHierarchyTrees* schrittweise von unten abgebaut. Dadurch wird sichergestellt, dass kein Ergebnis etwa durch eine Zusammenfassung weiter oben im Baum verloren geht. Beispiel 19 verdeutlicht diesen Vorgang.

Beispiel 19 Schrittweiser Abbau des Baums durch das Merging



Im ersten Schritt wird Knoten 4.1 als tiefster Knoten ausgewählt. Es wird nun über die Knoten 3.2, 4.1 und 4.2 entschieden. Die Entscheidung ist *MERGE_DOWN*, also wird der Vaterknoten als nicht im Ergebnis markiert, die Kindknoten 4.1 und 4.2 aus dem Baum entfernt und der Ergebnisliste hinzugefügt.

Als nächstes wird der Knoten 3.1 als tiefster Knoten ausgewählt, das bedeutet es wird über die Knoten 2.2, 3.1 und 3.2 entschieden. Diesmal ist die Entscheidung *DO_NOT_MERGE*. Jetzt werden nur die Kindknoten aus dem Baum entfernt und der Ergebnisliste hinzugefügt. Da allerdings Knoten 3.2 im vorherigen Schritt markiert wurde, wird letztendlich nur Knoten 3.1 der Ergebnisliste hinzugefügt.

Im letzten Schritt wird über die Knoten 1, 2.1, 2.2 und 2.3 entschieden. Es wird mit *MERGE_UP* entschieden und damit werden nur die Kindknoten aus dem Baum entfernt, aber nicht der Ergebnisliste hinzugefügt.

Dann ist nur noch der Wurzelknoten des Baums übrig und wird der Ergebnisliste hinzugefügt, da er nicht vorher markiert wurde. Damit sind die Knoten 1, 3.1, 4.1 und 4.2 in der Ergebnisliste enthalten.

Wenn alle Dokumente bearbeitet sind, werden in der Methode *generateHitList()* die Ergebnisse der Zusammenfassung aus jedem *HitHierarchyTree* eingesammelt (*getDocumentResults()*).

Merge (NoMerge und DefaultMerge)

Das Interface Merge besitzt nur die Methode *decideMerge()*. Dort wird die Entscheidung darüber getroffen, ob ein Vaterknoten aus den Ergebnissen gestrichen wird (siehe Abschnitt 4.5.3), des-

sen Kindknoten gestrichen werden (siehe Abschnitt 4.5.2) oder keine Aktion durchgeführt wird (siehe Abschnitt 4.5.4). Eine Implementierung dieser Klasse muss eine der drei Konstanten *MERGE_UP*, *MERGE_DOWN* oder *DO_NOT_MERGE* zurückliefern.

Es existieren die beiden Implemetierungen *NoMerge* und *DefaultMerge* (siehe Abbildung 19). Der Name der *Merge*-Klasse, die verwendet werden soll, kann der Klasse *XMLSearcher* im Konstruktor übergeben werden. Dadurch wird es möglich, die verschiedenen konkreten Implementierungen auszutauschen.

Die Klasse *NoMerge* liefert für beliebige Eingaben stets die Entscheidung *DO_NOT_MERGE* zurück. Damit lässt sich also die Zusammenfassung der Ergebnisse ausschalten. Diese Klasse wurde während der Entwicklung hauptsächlich zu Testzwecken verwendet.

Die Klasse *DefaultMerge* bearbeitet die Eingaben entsprechend den Ausführungen aus Abschnitt 4.5. Die Schwellenwerte $\text{thresh}_{\text{length}}$, $\text{thresh}_{\text{sim}}$ und $\text{thresh}_{\text{length}\cdot\text{sim}}$ sind in Variablen gespeichert und können im Code oder durch abgeleitete Klassen geändert werden.

5.2.5 Weitere Klassen

Hier werden Klassen aufgeführt, die von vielen der bereits beschriebenen Klassen benutzt werden, um mit ihnen Operationen auszuführen.

XMLSearchUtilities

Die Klasse *XMLSearchUtilities* stellt verschiedene statische Hilfsmethoden zur Verfügung, die von den in Abschnitt 5.2.2 bis Abschnitt 5.2.4 vorgestellten Klassen verwendet werden.

XPathLocation

XPathLocation repräsentiert ein Wertepaar, bestehend aus einem Markup-Element und dessen Position.

5.3 Benutzerschnittstelle

Um die Indexierung zu steuern wurde ein Kommandozeilenclient implementiert. Zum Zugriff auf die Suchfunktionalität wurden zwei Benutzerschnittstellen erstellt, ein Kommandozeilenclient und ein Webclient. Wir werden nun die Bedienung dieser Benutzerschnittstellen beschreiben.

5.3.1 Kommandozeilenclient für die Indexierung

Die Klasse *IndexXMLFiles* aus dem Paket *demo* implementiert einen Kommandozeilenclient, der die Indexierung steuert. Der Aufruf unterstützt drei Argumente:

java demo.IndexXMLFiles [-s <stop_word_file>] [-i <index_directory>] <root_directory>|-

Im Einzelnen sind dies:

- Das Wurzelverzeichnis der zu indexierenden Dokumente. Innerhalb dieses und der darunterliegenden Verzeichnisse wird jede Datei indexiert, unabhängig von der Dateiendung. Die Indexierung überspringt Dateien, die keine XML-Dokumente sind. Dieses Argument muss als

letztes ohne speziellen Schalter angegeben werden. Wird anstelle des Wurzelverzeichnisses „-“ angegeben, liest *IndexXMLFiles* die Dateinamen aus der Standardeingabe.

- Das Zielverzeichnis für den Index. Innerhalb dieses Verzeichnisses wird der Index angelegt. Dieses Argument wird mit dem Schalter „-i <*index_directory*>“ gesteuert und ist optional. Wird es weggelassen, wird der Index in einem Verzeichnis *index* im aktuellen Kontextpfad angelegt.

- Der Pfad und Dateiname zu einer Textdatei, die durch Zeilenumbrüche getrennt eine Stoppwortliste enthält. Dieses Argument wird mit dem Schalter „-s <*stop_word_file*>“ gesteuert und ist ebenfalls optional. Wird hier keine Angabe gemacht, so werden die in der Klasse *org.apache.lucene.analysis.StopAnalyzer* definierten Stoppwörter für die englische Sprache verwendet.

5.3.2 Kommandozeilenclient für die Suche

Der Kommandozeilenclient ist in der Klasse *demo.SearchFiles* implementiert. Er unterstützt im Wesentlichen zwei Modi zur Suche und einige Kommandozeilenoptionen.

Interaktiver Modus

Im interaktiven Modus (siehe Beispiel 20) wird jeweils die Eingabe einer Kategorie und einer Anfrage für die Kategorie erwartet. Dann folgt eine Abfrage, ob eine weitere Anfrage eingegeben werden soll, so dass sich eine beliebige Anzahl von Anfragen und deren Kategorien zur Suche hinzufügen lässt. Wird die Abfrage mit „*n*“ beantwortet, startet die Suche und präsentiert nach kurzer Zeit die Ergebnisse. Im interaktiven Modus wird zunächst eine bestimmte Anzahl Ergebnisse angezeigt. Über eine Abfrage lassen sich weitere Ergebnisse anzeigen.

Beispiel 20 **Suche im interaktiven Modus**

```
Category: Computer
Query: linux kernel
Add another category/query (y/n) ? n

0. - 100% - (1.6410634517669678) - Remote-Serial-Console-HOWTO.xml
- /book[1]/chapter[2]/section[2]/section[1]/table[1]
/tgroup[1]/thead[1]/row[1]/entry[2] - 14 - [Computer]

1. - 79% - (1.3128507137298584) - Remote-Serial-Console-HOWTO.xml
- /book[1]/chapter[2]/section[2]/section[1]/table[1]
/tgroup[1]/thead[1]/row[1]/entry[3] - 35 - [Computer]

2. - 70% - (1.1604070663452148) - Remote-Serial-Console-HOWTO.xml
- /book[1]/chapter[2]/section[2]/section[1]/table[1]/tgroup[1]
/thead[1] - 74 - [Computer]

[...]

Display more (y/n) ? n
```

Batch-Modus

Im Batch-Modus erwartet der Client nach einem Aufruf nur die Eingabe aller Anfragen und Kategorien in einer speziellen Anfragesyntax. Gültige Anfragen müssen der folgenden Grammatik genügen:

Syntax 1 **BNF für die Anfragesprache des Batch-Modus**

```
<query-sequence> ::= <query> { " " <query> }
<query>          ::= <category> <lucene-query>
<category>       ::= [ <category-string> ] [ <weight> ]
<weight>         ::= ":" <integer>
<lucene-query>   ::= "{" <lucene-query-string> "}"
```

Dabei ist <integer> eine natürliche Zahl, <category-string> der Kategorie- beziehungsweise Elementname und <lucene-query-string> die Anfrage für diese Kategorie.

Die Anfrage „*Computer:5{kernel} {debugging}*“ beispielsweise sucht in der Kategorie „*Computer*“ nach „*kernel*“ und ohne Kategorie nach „*debugging*“. Dabei wird die Kategorie Computer mit der fünffachen Gewichtung berücksichtigt.

Das Programm terminiert nach der Ausgabe einer bestimmten Anzahl Ergebnisse.

Aufrufparameter

Der Kommandozeilenclient wird folgendermaßen aufgerufen (alle Parameter sind optional):

java demo.SearchFiles -fmpxIX -i <index-directory> -R <rdf-file-location> <rdf-namespace> -n<number> -P <relative-directory>

Dabei ist zu unterscheiden zwischen der normalen Ergebnisanzeige und der Ausgabe im XML-Format. Zunächst beschreiben wir jedoch die allgemeinen Kommandoparameter, die von beiden Ausgabeformaten unterstützt werden:

- **-I** Standardmäßig startet der Client im Batch-Modus. Ist dieser Parameter gesetzt, startet der Client im interaktiven Modus.
- **-i <index-directory>** Der Index, der sich im Verzeichnis *index-directory* befindet, wird für die Suche verwendet.
- **-R <rdf-file-location> <rdf-namespace>** Für den ERG werden anstelle der Standardwerte die durch *rdf-file-location* angegebenen Dateien und der durch *rdf-namespace* spezifizierte Namensraum verwendet.
- **-m** Es erfolgt kein Zusammenfassen der Ergebnisse. Genauer gesagt wird zur Zusammenfassung die Klasse *search.NoMerge* (siehe Abschnitt 5.2.4) verwendet.
- **-X** Die Ergebnisse werden nicht nach Relevanz sortiert ausgegeben, sondern sortiert nach Dateiname und XPath.
- **-n<number>** Die Ausgabe zeigt initial *number* Ergebnisse an. Im interaktiven Modus können jedoch weitere Ergebnisse angezeigt werden.

- **-P <relative-directory>** Dieser Parameter erlaubt es, den Pfad für die Ergebnisdokumente nur ab einer gewissen Ebene anzuzeigen. Die Angabe `-P "C:\XMLRetrieval\xml\"` ergibt für ein Ergebnisdokument C:\XMLRetrieval\xml\mu\2000\u1075.xml die Ausgabe mu\2000\u1075.xml.
- **-x** Ist dieser Parameter gesetzt, erfolgt die Ausgabe der Ergebnisse in einem XML-Format, das dem INEX-Submission-Format ähnelt (siehe Beispiel 21). Andere Ausgaben werden unterdrückt. Dateinamen werden ohne die Endung „*.xml*“ angezeigt.

Die folgenden Parameter werden bei der Ausgabe im XML-Format nicht interpretiert. Sie gelten nur für die normale Ergebnisausgabe:

- **-f** Durch das Setzen dieses Parameters wird nur der Dateiname des Dokumentes ausgegeben, in dem der Treffer gefunden wurde.
- **-p** Mit diesem Parameter können die Inhalte der jeweiligen Ergebnisfragmente ausgegeben werden. Da dazu die Dateien einzeln eingelesen werden kann die Ausgabe mit diesem Parameter insbesondere bei längeren Ergebnislisten etwas länger dauern.

Beispiel 21 **Ausgabe mit gesetzten Parametern -x und -P**

```
<?xml version='1.0'?>
<topic>
  <result>
    <file>DB2-HOWTO</file>
    <path>/article[1]/sect1[9]/sect2[2]/qandaset[1]</path>
    <rsv>0.9523659348487854</rsv>
  </result>
  <result>
    <file>Kernel-HOWTO</file>
    <path>/article[1]/sect1[3]/para[5]</path>
    <rsv>0.9483765959739685</rsv>
  </result>
[...]
</topic>
```

5.3.3 Webclient für die Suche

Mittels des Target „*make-war*“ aus dem Ant-Buildfile „build.xml“ lässt sich ein Webarchiv erstellen, das beispielsweise in Apache Tomcat installiert werden kann. Der Webclient bietet etwas weniger Funktionalität als der Kommandozeilenclient. Abbildung 21 zeigt die Suchmaske.

Es gibt für jede Kategorie beziehungsweise Anfrage drei Auswahlfelder. Die Kategorien können aus einer Drop-Down-Liste, die aus dem ERG generiert wird, ausgewählt werden. Optional kann eine Gewichtung mit angegeben werden. In dem Feld darunter wird die Anfrage eingegeben. Mit einem Klick auf „*Add another category*“ lassen sich beliebig viele Eingabefelder hinzufügen.

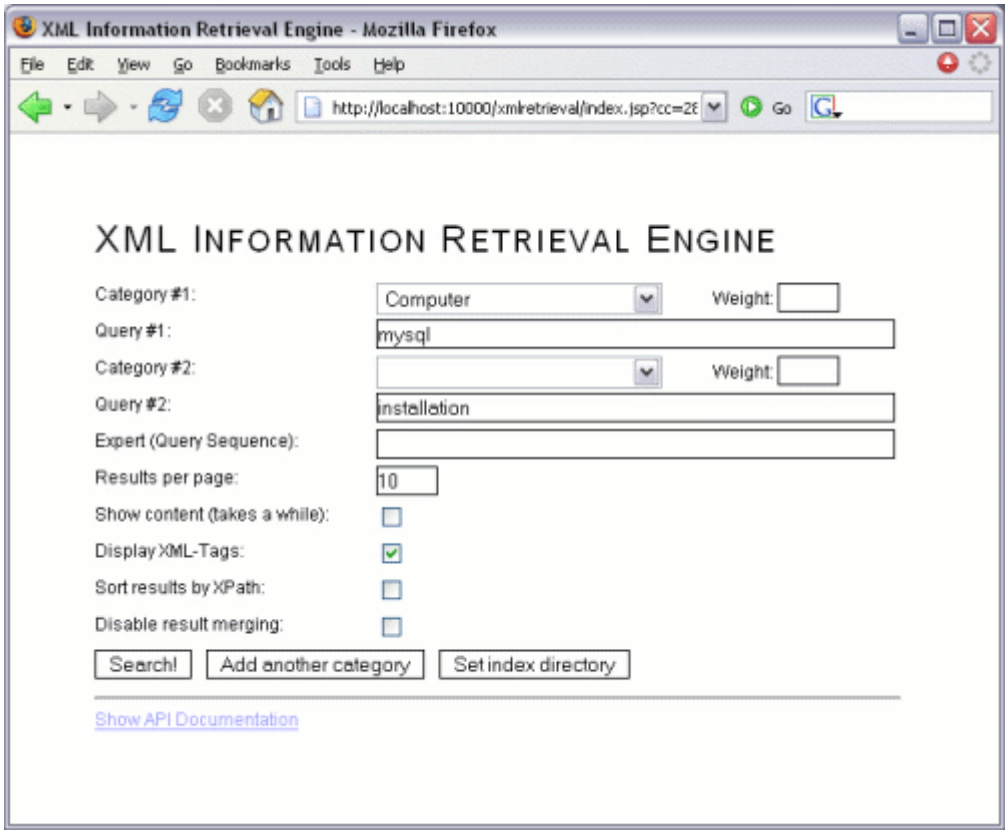
Alternativ kann eine Query Sequence nach Syntax 1 im *Expert*-Feld eingetragen werden. Auf diese Art werden auch Elementnamen als Kategorie unterstützt.

Außerdem kann die Anzahl der Ergebnisse pro Seite eingestellt werden. Als zusätzliche Optionen kann

- der Inhalt auf der Ergebnisseite angezeigt werden („*Show content*“),
- die XML-Tags bei der Inhaltsanzeige ein- beziehungsweise ausgeschaltet werden („*Display XML-Tags*“),
- die Ergebnisse wahlweise nach Relevanz oder nach XPath sortiert werden („*Sort results by XPath*“) und
- das Zusammenfassen der Ergebnisse abgeschaltet werden („*Disable result merging*“).

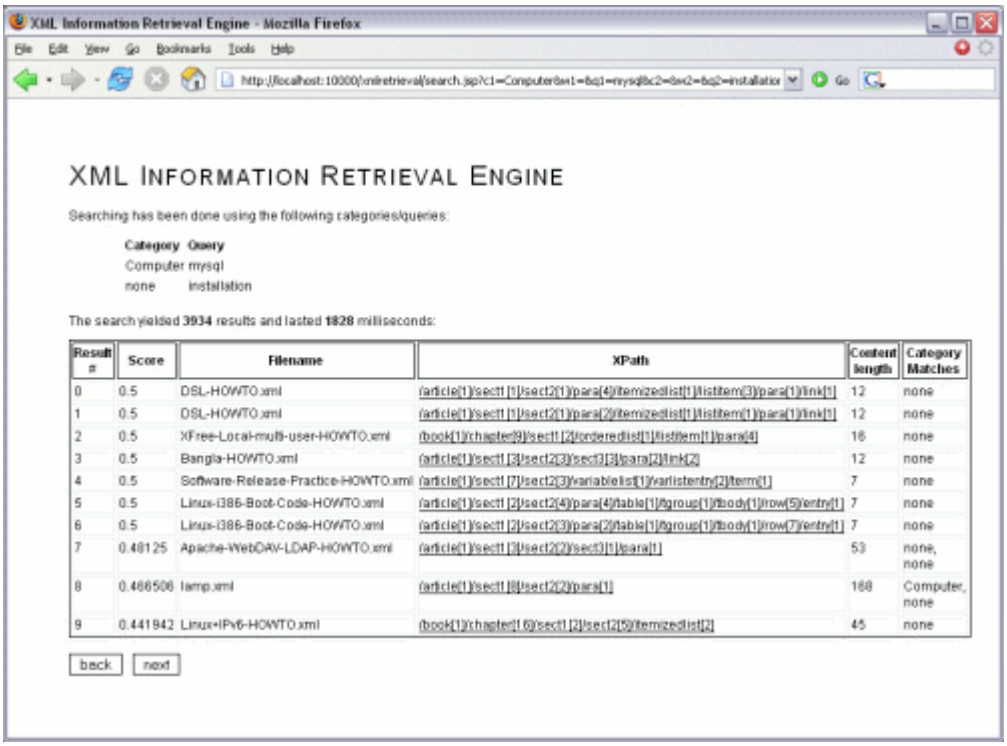
Der Button „*Set index directory*“ erlaubt es, das Indexverzeichnis während der Laufzeit zu bestimmen. Der eingetragene Wert wird dann in einem Session-Cookie gespeichert und muss vor der ersten Suche angegeben werden.

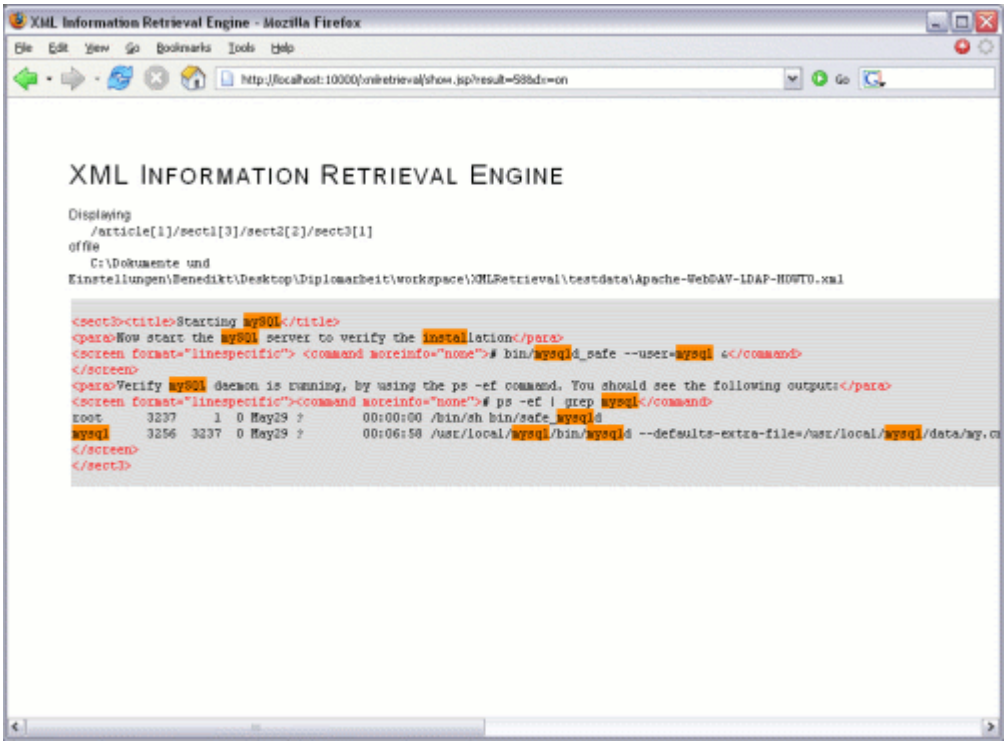
Abbildung 21 Suchfenster im Webclient



Ein Klick auf „Search!“ startet die Suche. Die Ergebnisse werden wie in Abbildung 22 oben dargestellt.

Abbildung 22 Ergebnisseite und Anzeige des Inhaltes eines Ergebnisfragments im Webclient





Von links nach rechts werden folgende Informationen Angezeigt: Nummer des Ergebnisses („*Result #*“), Relevanzwert („*Score*“), Dateiname („*Filename*“), XPath, Länge des Inhaltes ohne XML-Tags in Zeichen („*Content length*“) und die Angabe, in welchen Kategorien dieses Dokument ein Treffer ist („*Category Matches*“). Durch einen weiteren Klick auf den XPath eines Ergebnisses wird dessen Inhalt wie in Abbildung 22 unten dargestellt. Dabei werden die XML-Tags zu besseren Lesbarkeit farbig markiert. Außerdem sind die Terme aus der Anfrage deutlich hervorgehoben.

Ist „*Show content*“ eingeschaltet, werden die Inhalte der Dokumentfragmente auf der Ergebnisseite zwischen „*Score*“ und „*Filename*“ eingeblendet.

Kapitel 6 Zusammenfassung und Bewertung

Das Gebiet des Information Retrieval auf XML-Dokumenten ist ein sehr neues Forschungsgebiet, das viele Herausforderungen bietet. Der aktuelle Stand der Forschung zeigt, dass viele Fragen noch immer nicht zufriedenstellend beantwortet sind. Die Probleme mit überlappenden Ergebnissen und der optimalen Ergebnisgranularität sind Kernprobleme, denen sich jeder Ansatz stellen muss. Sinnvoll erscheint auch der Versuch, die Metainformationen der Dokumente zur Verbesserung des Retrieval einzusetzen. Dabei hat sich noch kein Konzept herausgestellt, das anderen Ansätzen offensichtlich überlegen ist.

Ein weiteres Problem ist aber auch die Unklarheit darüber, ob die beim INEX-Workshop verwendeten Bewertungsmaßstäbe wirklich dazu geeignet sind, verschiedene Forschungsansätze miteinander zu vergleichen.

6.1 Zusammenfassung

In dieser Arbeit haben wir damit begonnen, in das Gebiet des traditionellen Information Retrieval einzuführen. Zunächst haben wir durch Begriffsklärungen und eine Themenabgrenzung die Grundlagen für das Verständnis der Thematik geschaffen. Darauf aufbauend haben wir die Funktionsweise von Indexierung und Suche mit verschiedenen gebräuchlichen Retrievalmodellen auf unstrukturierten Dokumenten beschrieben. Es folgte eine Erklärung, wie sich Forschungsansätze nach deren Effizienz aber vor allem Effektivität anhand der Maße Precision und Recall bewerten und miteinander vergleichen lassen.

Danach sind wir den Schritt von unstrukturierten Dokumenten hin zu semistrukturierten XML-Dokumenten gegangen, beginnend mit einer kurzen Einführung in XML. Wir haben einige wichtige Probleme angesprochen, die sich im Zusammenhang mit der Verwendung von XML-Dokumenten ergeben, wie zum Beispiel die Notwendigkeit der Erweiterung der Indexstrukturen. Die beiden wichtigen Punkte, wie sich das Wissen über die Struktur der Dokumente auf die Berechnung der Relevanz auswirkt und in welcher Granularität die Ergebnisse zurückgeliefert werden sollen, wurden einführend behandelt. Wie bereits erwähnt gibt es für diese Probleme noch keine Standardlösungen. Der INEX-Workshop versucht durch eine große Dokumentenkollektion und entsprechende Topics eine Basis für die Forschung zu bilden, anhand derer sich Forschungsergebnisse auch und vor allem im Vergleich mit anderen Ansätzen bewerten lassen. Dazu gehört auch die Verwendung geeigneter Bewertungsmaßstäbe, die auf die Besonderheiten von mit XML

ausgezeichneten Dokumenten angepasst sind. Die Forschungsansätze unterscheiden sich zum Teil sehr stark. Daher haben wir versucht, eine Kategorisierung nach verschiedenen Kriterien einzuführen und anhand der beiden Systeme HyREX und XXL die Vielfalt der Lösungsansätze aufzuzeigen.

Die Implementierung einer XML-Suchmaschine stand im Zentrum dieser Arbeit. Daher haben wir zunächst das Kernkonzept, die Verwendung von Element Relationships zur Verbesserung des Retrieval, vorgestellt. Dabei werden durch Beziehungen der Markup-Elementnamen, die in einem Graph gespeichert sind, gewisse Suchergebnisse bevorzugt. Die Implementierung dieses Konzeptes fand in Java auf der Basis der Suchmaschinenbibliothek Lucene statt. Wir haben eine Variante der Indexierung vorgestellt, die es uns erlaubt, mit Lucene auf XML-Dokumentfragmenten zu suchen. Dann haben wir erläutert, wie die Element Relationships in der Suche verwendet werden, indem alle ein bestimmtes Markup-Element umschließenden Ergebnisse mit einem Kohärenzwert multipliziert werden. Ein wichtiger Teil einer XML-Suchmaschine ist die Zusammenfassung überlappender Ergebnisse, die nötig ist, damit sich in den Ergebnissen nicht zu viel Redundanz befindet. Auch hier haben wir ein Konzept vorgeschlagen, das die Problematik behandelt.

Abschließend folgte eine detailierte Beschreibung der Implementierung und der Benutzerschnittstellen, um weitere Forschungen mit der in dieser Arbeit entwickelten Suchmaschine zu ermöglichen.

6.2 Bewertung

Der nächste logische Schritt wäre eine Bewertung unserer Suchmaschine anhand der in Kapitel 3 angesprochenen Maßstäbe. Bedauerlicherweise ist dies mit der von uns verwendeten Dokumentenkollektion aus mehreren Gründen nicht sinnvoll möglich.

Unsere Dokumentenkollektion, die Linux-HOWTOs, umfasst nur knapp 200 Dokumente. Hinzu kommt außerdem, dass sich die einzelnen Dokumente immer sehr stark auf ein einzelnes Thema konzentrieren. Eine repräsentative Dokumentenkollektion sollte zum Einen aus erheblich mehr Dokumenten bestehen; die beim INEX-Workshop verwendete Kollektion besteht aus mehr als 12000 Artikeln. Zum Anderen sollten sich in der Kollektion immer mehrere Dokumente befinden, die ein bestimmtes Themengebiet mehr oder weniger ausführlich behandeln. Das ist notwendig, damit für eine bestimmte Anfrage auch genügend unterschiedliche relevante Ergebnisse in der Kollektion vorhanden sind.

Ein weiteres mit den obigen Erklärungen zusammenhängendes Problem ist die Entwicklung von geeigneten Topics. Damit eine objektive Beurteilung eines Retrieval-Systems möglich ist, muss die Effektivität über eine größere Anzahl von unterschiedlichen Topics gemittelt werden [KaLM03]. Bereits die Erstellung solcher Topics stellt ein Problem dar. Abgesehen von den wenigen Möglichkeiten, die eine derart kleine Dokumentenkollektion bietet, sollten Topics möglichst unterschiedliche, vom Umfang teils sehr speziell, teils allgemeinere Themen behandeln und sinnvollerweise von Experten auf dem jeweiligen Gebiet des Topics erstellt werden. Nach der Erstellung der Topics ergibt sich ein weiteres Problem, das auch andere Testkollektionen betrifft: Die Ermittlung der zu einem Topic in einer Dokumentenkollektion insgesamt enthaltenen relevanten Ergebnisse, auf der die Berechnung des Recall (siehe Abschnitt 2.6.2) basiert. Das setzt wiederum eine intensive Untersuchung der Dokumentenkollektion voraus.

Man sieht also, dass die Bewertung einer XML-Suchmaschine kein triviales Problem darstellt. Daher werden wir uns auf einige andere Aspekte beschränken und die Evaluierung der Effektivität unseres Ansatzes anhand der beschriebenen Bewertungsmaßstäbe zunächst zurückstellen.

Wir beginnen mit einer Abschätzung der Laufzeit und des Speicherbedarfs der Indexierung. Dann werden wir kurz auf die Effizienz der Suche eingehen. Schließlich werden wir versuchen, durch vergleichende Betrachtungen einige Aussagen zum Nutzen unseres Kernkonzeptes, den Element Relationships, zu machen.

6.2.1 Laufzeit und Speicherplatzbedarf der Indexierung

Für die Indexierung spielen zwei wichtige Faktoren eine Rolle. Der wichtigste Faktor ist der vom Index benötigte Speicherplatz. Das ist nicht nur in Bezug auf die Ressourcen ein zentrales Element, sondern wirkt sich auch unmittelbar auf die Suche aus (siehe Abschnitt 6.2.2). Der zweite Faktor ist die Laufzeit der Indexierung. Eine Indexierung findet zwar vorab statt und der Zeitbedarf ist für den laufenden Betrieb einer Suchmaschine nur von untergeordneter Wichtigkeit. Trotzdem sollte auch hier ein erträgliches Maß eingehalten werden.

Die Anzahl der pro Dokument zu indexierenden Dokumentfragmente ist abhängig von der Verschachtelungstiefe, das heißt je tiefer die Dokumente ausgezeichnet sind, desto mehr Fragmente müssen indexiert werden. Die Anzahl der pro Dokument zu indexierenden Terme ist wiederum abhängig davon, wie weit die inneren Elemente die äußeren überdecken. In Beispiel 22 haben beide Dokumentfragmente eine Verschachtelungstiefe von 2. Durch die geringe Überdeckung des „*em*“-Elements im ersten Fragment müssen jedoch viel weniger Terme indexiert werden als im zweiten Fragment. Das zweite Fragment zeigt jedoch einen Extremfall, der nur äußerst selten auftreten sollte.

Beispiel 22 **Überdeckung der äußeren Elemente durch die inneren (aus DHCP.xml)**

```
<sec>
    Feedback is <em>most certaintly</em> welcome for this
    document.
</sec>

<sec>
    <para>
        Feedback is most certaintly welcome for
        this document.
    </para>
</sec>
```

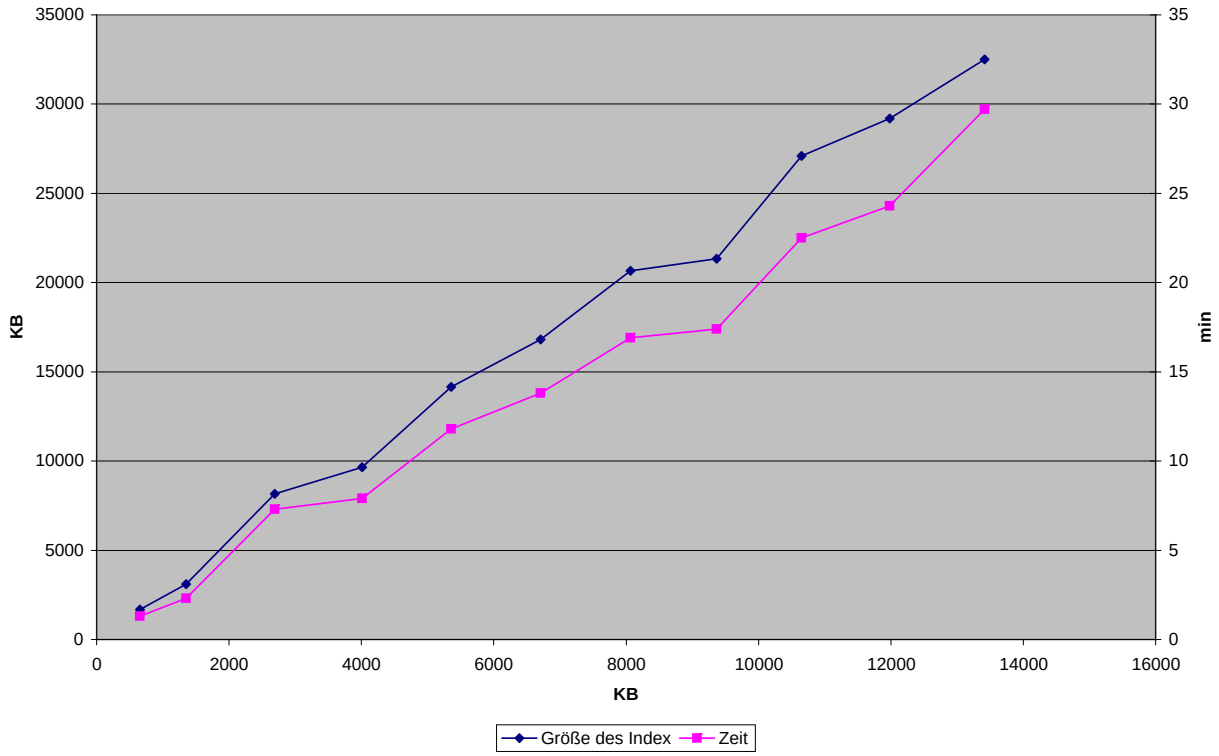
Die Werte aus Abbildung 23 wurden auf einem Pentium 4 2,4 GHz mit 768 MB RAM ermittelt. Man sieht, dass der Speicherplatzbedarf linear zur größe der Dokumentenkollektion steigt. Der Index ist damit ungefähr um einen Faktor 3 bis 4 größer als die Dokumentenkollektion. Diese Steigerung ist dadurch zu erklären, dass im Gegensatz zu traditionellen Dokumenten bei XML-Dokumenten eine größere Menge an Informationen gespeichert werden müssen. Da die von uns verwendete Dokumentenkollektion mit vergleichsweise vielen Elementen ausgezeichnet ist und

damit bereits einen Extremfall darstellt, erwarten wir für andere Dokumentenkollektionen keine Verschlechterung in Bezug auf die Größe des Index.

Wie aus Abbildung 23 zu erkennen ist, steigt auch der Zeitbedarf linear an. Für unsere Dokumentenkollektion von 202 Dokumenten in 13,1 MB benötigte die Indexierung knapp eine halbe Stunde. Da wie bereits erwähnt die Indexierung nur im Vorfeld stattfindet, ist dieses Laufzeitverhalten akzeptabel.

Erste Tests mit der INEX-Testkollektion (494 MB) brachten eine Indexgröße von etwa 2 GB bei einer Laufzeit von 10 Stunden. Damit scheint sich das Verhalten auch bei einer erheblich größeren Dokumentenkollektion zu bestätigen.

Abbildung 23 Speicherplatzbedarf und Laufzeit der Indexierung



6.2.2 Effizienzbetrachtung der Suche

Die Geschwindigkeit der Suche hängt vor allem von der Anzahl der Ergebnisse ab, die Lucene im Index findet. Unser Konzept sieht vor, alle Ergebnisse in den Hauptspeicher zu laden und dann weiterzubearbeiten. Daher steigt der Speicherbedarf proportional zu der Anzahl der Ergebnisse an, scheint aber selbst bei einer sehr großen Anzahl Ergebnisse noch handhabbar zu sein.

Der größte Geschwindigkeitsverlust entsteht an der Schnittstelle zwischen Lucene und unserer Implementierung. Das Einlesen der Elemente aus dem Index in die von uns verwendeten Speicherstrukturen kostet erheblich Laufzeit.

Bei der DocBook-Dokumentenkollektion mit einer durchschnittlichen Anzahl Ergebnisse im Bereich von 3000 bis 10000 Ergebnissen liegt die Reaktionszeit im Bereich weniger Sekunden. Erste Tests mit der erheblich umfangreicheren INEX-Dokumentenkollektion lieferten teilweise mehrere hunderttausend Ergebnisse. Dadurch stieg die für die Bearbeitung benötigte Zeit auf 3 bis 5 Minuten an.

6.2.3 Bewertung der Element Relationships

Die Bewertung der Element Relationships stellt uns zunächst vor ähnliche Probleme, auf die wir auch wie Eingangs erwähnt bei der Bewertung der Effektivität stoßen. Daher ist dieser Abschnitt eher als Beispiel zu betrachten, wie mit einer genügend großen Dokumentenkollektion und einer ausreichenden Anzahl an Topics eine solche Bewertung vorgenommen werden kann.

Die Bewertung erfolgt anhand verschiedener Kriterien bei Betrachtung der ersten 15 Ergebnisse. Wir bewerten die Ergebnisse hier für eine einzelne Anfrage danach, ob sie relevant oder nicht relevant in Bezug auf die Anfrage sind. Die Betrachtung einer größeren Anzahl Ergebnisse auch mit komplexeren Bewertungskriterien wie in Abschnitt 3.4.2 ist jedoch empfehlenswert. Wir vereinfachen an dieser Stelle, da aufgrund der zu Beginn von Abschnitt 6.2 angesprochenen Probleme kein aussagekräftiges Ergebnis möglich ist.

Wir betrachten die Ergebnismenge R für eine Anfrage ohne Angabe einer Kategorie vergleichend mit der Ergebnismenge R_{cat} für dieselbe Anfrage mit Angabe einer geeigneten Kategorie. Dabei sind verschiedene Werte dazu geeignet, eine Aussage über den Nutzwert der Element Relationships zu machen:

1. $\frac{precision(R_{cat})}{precision(R)}$, die relative Steigerung der Precision,
2. $\frac{overlap(R_{cat})}{overlap(R)}$, die relative Steigerung des Overlap Indicator (siehe Abschnitt 3.4.2) und
3. $\frac{\{(r \in R) | (r \text{ ist relevant} \wedge r \notin R_{cat})\}}{|R|}$, die relative Anzahl der weggefallenen relevanten Ergebnisse,
4. $\frac{\{(r \in R_{cat}) | (r \text{ ist relevant} \wedge r \notin R)\}}{|R|}$, die relative Anzahl der neu hinzugekommenen relevanten Ergebnisse.

Beispiel 23 **Bewertung der Element Relationships**

Wir betrachten die Beispielanfrage „bash set path“ in der Kategorie „Things“.

	Anfrage: bash set path Kategorie: keine	Anfrage: bash set path Kategorie: Things
Relevanz der Ergebnisse	1 0 1 0 0 0 0 1 0 0 0 0 0 0 0	1 0 0 1 1 0 0 0 0 1 0 1 1 0 0
Precision	0,2	0,4
Overlap Indicator	0,2	0,4

Damit ergeben sich für die obigen Formeln folgende Werte:

$$\frac{precision(\mathrm{R}_{\mathrm{Things}})}{precision(\mathrm{R})} = \frac{0{,}4}{0{,}2} = 2\,,\ \frac{overlap(\mathrm{R}_{\mathrm{Things}})}{overlap(\mathrm{R})} = \frac{0{,}4}{0{,}2} = 2$$

$$\frac{\{(\mathrm{r} \in \mathrm{R}) | (\mathrm{r\ ist\ relevant} \wedge \mathrm{r} \notin \mathrm{R}_{\mathrm{Things}})\}}{|\mathrm{R}|} = \frac{0}{15} = 0$$

$$\frac{\{(\mathrm{r} \in \mathrm{R}_{\mathrm{Things}}) | (\mathrm{r\ ist\ relevant} \wedge \mathrm{r} \notin \mathrm{R})\}}{|\mathrm{R}|} = \frac{3}{15} = 0{,}2$$

Das bedeutet, dass die Verwendung von Element Relationships in diesem Fall eine Verbesserung der Precision bewirkt, allerdings bei gleichzeitiger Steigerung des Overlap Indicator. Positiv zu bewerten ist, dass in diesem Fall unter Verwendung einer Kategorie keine relevanten Ergebnisse wegfallen.

Sollen durch die Anfrage „bash set path“ Informationen, wie man beim Bash-Kommandozeileninterpreter den Pfad setzt, gefunden werden, könnte die Anfrage „bash“ in der Kategorie „Things“ zusammen mit „set path“ ohne Kategorie auch eine Variante darstellen, da höchstwahrscheinlich nur „bash“ innerhalb der Elemente aus der Kategorie „Things“ auftritt. Der Unterschied dieser beiden Anfragen ist, dass für die erste Variante zunächst nur eine Lucene-Suche durchgeführt wird und dann die Ergebnisse, die eines der Elemente der Kategorie „Things“ enthalten, bevorzugt werden. Bei der zweiten Variante werden zwei Lucene-Suchen durchgeführt, die nach der Berechnung der

Element Relationships miteinander kombiniert werden. In diesem Fall führt dies allerdings nicht zu einer Verbesserung gegenüber der Verwendung nur einer Kategorie:

	Anfrage: bash set path Kategorie: Things	Anfrage: bash Kategorie: Things Anfrage: set path Kategorie: keine
Relevanz der Ergebnisse	1 0 0 1 1 0 0 0 0 1 0 1 1 0 0	1 0 1 0 0 0 1 0 0 0 1 0 0 0 0
Precision	0,4	0,27
Overlap Indicator	0,4	0,33

Als nächstes betrachten wir die Anfrage „install mysql“ in der Kategorie „Computer“.

	Anfrage: install mysql Kategorie: keine	**Anfrage: install mysql Kategorie: Things**
Relevanz der Ergebnisse	0 0 0 0 0 1 0 0 0 0 0 0 0 0 0	0 0 1 1 0 0 0 0 0 1 1 0 0 0 0
Precision	0,07	0,27
Overlap Indicator	0,33	0,4

Damit ergibt sich:

$$\frac{precision(\mathrm{R_{Computer}})}{precision(\mathrm{R})} = \frac{0{,}27}{0{,}07} = 3,86\,,\ \frac{overlap(\mathrm{R_{Computer}})}{overlap(\mathrm{R})} = \frac{0{,}4}{0{,}33} = 1,21$$

$$\frac{\{(r \in \mathrm{R}) | (r \text{ ist relevant} \wedge r \notin \mathrm{R_{Computer}})\}}{|\mathrm{R}|} = \frac{0}{15} = 0$$

$$\frac{\{(r \in \mathrm{R_{Computer}}) | (r \text{ ist relevant} \wedge r \notin \mathrm{R})\}}{|\mathrm{R}|} = \frac{3}{15} = 0{,}2$$

Auch in diesem Beispiel lässt sich durch die Angabe einer Kategorie das Ergebnis wieder deutlich verbessern. Der Anstieg des Overlap Indicator ist nicht so drastisch wie im ersten Beispiel, aber dennoch recht hoch. Es sind wieder keine relevanten Ergebnisse weggefallen.

Die Beispiele zeigen bereits, dass eine Verbesserung des Retrieval durch die Verwendung von Element Relationships zumindest im Einzelfall möglich ist. Allerdings sollte diesen Ergebnissen noch nicht zu viel Bedeutung beigemessen werden. Eine objektive Beurteilung der Element Relationships kann erst durch entsprechende Tests mit einer genügend großen Dokumentenkollektion erfolgen.

6.3 Ausblick

In der Implementierung gibt es einige Stellen, an denen Erweiterungen sinnvoll sind. Zunächst ist hier die Zusammenfassung der Ergebnisse zu nennen, wo durch das *Merge*-Interface die Voraussetzungen für eine Erweiterung bereits gegeben sind. Andere Konzepte für die Zusammenfassung können an dieser Stelle implementiert werden.

Ein großer Problembereich ist die Schnittstelle zu Lucene. Hier sind verschiedene Varianten der Erweiterung denkbar. Ein Ansatz ist die Verwendung einer anderen Suchmaschine als Basis, was durch die von Lucene getrennte Verarbeitung der Ergebnisse ermöglicht wird. Sinnvoll wäre aber auch ein weitreichender Eingriff in Lucene, um die Indexstrukturen und den Abruf der Ergebnisse besser auf unseren Ansatz anzupassen. Hier wäre beispielsweise denkbar, die Fragmente bereits sortiert nach Dokument zurückzuliefern, um die restliche Verarbeitung dokumentweise durchführen zukönnen. Das würde zu einer starken Entlastung des Hauptspeicherbedarfs führen. Dadurch wären dann auch die Grundlagen für eine verteilte Ausführung der Suche geschaffen. Die Suche könnte zunächst für Indices verschiedener Teile der Dokumentenkollektion auf unterschiedlichen Rechnern stattfinden und die Ergebnisse nach Abschluss der Zusammenfassung wieder zusammenführen.

Oberste Priorität hat aber momentan eine Evaluierung unseres Retrievalsystems auf dem INEX-Workshop 2005. Dazu muss zunächst noch ein ERG für das bei der INEX-Dokumentenkollektion verwendete Schema erstellt werden, wobei auch das Verfahren für die Erstellung eines ERG und die Bestimmung geeigneter Kohärenzwerte noch nicht vollständig geklärt ist. Leider ist das in der INEX-Dokumentenkollektion verwendete Markup nicht mit besonders vielen Inline-Elementen ausgestattet, so dass bereits damit zu rechnen ist, dass die Element Relationships nicht wirklich ausgenutzt werden können. Um sinnvolle Erkenntnisse in Bezug auf die Element Relationships zu gewinnen wird eine umfangreiche Dokumentenkollektion benötigt, die mit ähnlich reichhaltigen Inline-Markup-Elementen ausgezeichnet ist wie die Linux-HOWTOs.

Literaturverzeichnis

[ApLu05] o.V.:
Apache Lucene
The Apache Software Foundation, 2005
(Verfügbar unter: http://lucene.apache.org/java/docs)

[AYB+04] Sihem Amer-Yahia, Chavdar Botev, Stephen Buxton, Pat Case, Jochen Doerre, Darin McBeath, Michael Rys, Jayavel Shanmugasundaram:
XQuery 1.0 and XPath 2.0 Full-Text
World Wide Web Consortium, 2004
(Verfügbar unter: http://www.w3.org/TR/xquery-full-text)

[BCF+04] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, Jérôme Siméon:
XQuery 1.0: An XML Query Language
World Wide Web Consortium, Oktober 2004
(Verfügbar unter: http://www.w3.org/TR/xquery)

[BYRN99] Ricardo Baeza-Yates und Berthier Ribeiro-Neto:
Modern Information Retrieval
Addison Wesley, Harlow, England, 1999

[CABF04] Emiran Curtmola, Sihem Amer-Yahia, Philip Brown, Mary Fernández:
GalaTex: A Conformant Implementation of the XQuery Full-Text Language
AT&T Technical Report TD-66VRF4, 19. November 2004.

[CaMS00] David Carmel, Yoelle Maarek, Aya Soffer:
XML and Information Retrieval: a SIGIR 2000 Workshop
In: *ACM SIGIR Forum, Band 34, Nummer 1, ACM Press, New York (NY), U.S.A., 2000,* S. 31–36,

[ClDe99] James Clark, Steve DeRose:
XML Path Language (XPath) Version 1.0
World Wide Web Consortium, November 1999
(Verfügbar unter: http://w3c.org/TR/xpath)

[Dopi05] Philipp Dopichaj:
Element Relationship: Exploiting Inline Markup for Better XML Retrieval
In: *Proceedings BTW (BTW, Karlsruhe 2. – 4. März), 2005*, S. 285–294

[FaWa04] David C. Fallside, Priscilla Walmsley:
XML Schema Part 0: Primer Second Edition
World Wide Web Consortium, 2004
(Verfügbar unter: http://w3c.org/TR/xmlschema-0)

[Ferb03] Reginald Ferber:
Information Retrieval - Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web
dpunkt.verlag, Heidelberg, 2003

[Fris87] M. E. Frisse:
Searching for information in a hypertext medical handbook
In: *Communications of the ACM, Band 31, Nummer 7, ACM Press, New York (NY), U.S.A., 1988,* S. 880 – 886

[FuGr01] Norbert Fuhr, Kai Großjohann:
XIRQL: A Query Language for Information Retrieval in XML Documents
In: *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New Orleans (LA), U.S.A., 2001,* S. 172–180

[Fuhr04] Norbert Fuhr:
Information Retrieval, Skriptum zur Vorlesung im SS 04
Universität Dortmund, Juni 2004

[Fuhr96] Norbert Fuhr:
Gesellschaft für Informatik (GI), Fachgruppe 2.5.4 bzw. 4.9.3, Information Retrieval
Universität Duisburg, Januar 1996
(Verfügbar unter:
http://www.uni-hildesheim.de/~fgir/pages/info.html)

[FuLa04] Norbert Fuhr, Mounia Lalmas:
Overview of the INitiative for the Evaluation of XML Retrieval (INEX) 2003
In: *INEX 2003 Workshop Proceedings, Dagstuhl, 15. - 17. Dezember, 2003*, S. 1–11

[GAFG02] Norbert Gövert, Mohammad Abolhassani, Norbert Fuhr, Kai Großjohann:
Content-oriented XML retrieval with HyREX
In: *Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval (INEX), Dagstuhl, 11. - 8. Dezember, 2002*, S. 26–32

[GKFL03] Norbert Gövert, Gabriella Kazai, Norbert Fuhr, Mounia Lalmas:
Evaluating the effectiveness of content-oriented XML retrieval
Technical report, Universität Dortmund, 2003

[Henr99] Andreas Henrich:
Information Retrieval
Foliensammlung, Universität Bamberg, 1999

[Hiem02] Djoerd Hiemstra:
A database approach to content-based XML retrieval
In: *Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval (INEX), Dagstuhl, 11. - 8. Dezember, 2002*, S. 111–118

[KaLM03] Gabriella Kazai, Mounia Lalmas, Saadia Malik:
INEX'03 Guidelines for Topic Development
In: *INEX 2003 Workshop Proceedings, Dagstuhl, 15. - 17. Dezember, 2003*, S. 192–199

[KaLP04] Gabriella Kazai, Mounia Lalmas, Benjamin Piwowarski:
INEX 2004 Relevance Assessment Guide
In: *INEX 2004 Workshop Pre-Proceedings, Dagstuhl, 6. - 8. Dezember, 2004*, S. 241–248

[KaLV04] Gabriella Kazai, Mounia Lalmas, Arjen P. de Vries:
The Overlap Problem in Content-Oriented XML Retrieval Evaluation
In: *Proceedings of the 27th annual international conference on Research and development in information retrieval, Sheffield, England, 2004*, S. 72–79

[KGSL03] Wayne Kelly, Shlomo Geva, Tony Sahama, Wengkai Loke:
Distributed XML Information Retrieval
In: *INEX 2003 Workshop Proceedings, Dagstuhl, 15. - 17. Dezember, 2003*, S. 126–133

[KiSo04] Heespo Kim, Heejung Son:
Kyungpook National University at INEX 2004: Interactive Track
In: *INEX 2004 Workshop Pre-Proceedings, Dagstuhl, 6. - 8. Dezember, 2004*, S. 204–210

[Kuhl90] Rainer Kuhlen:
Zum Stand pragmatischer Forschung in der Informationswissenschaft.
In: *J. Herget and R. Kuhlen, Herausgeber, Pragmatische Aspekte beim Entwurf und Betrieb von Informationssystemen. Proceedings des 1. Internationalen Symposiums für Informationswissenschaft, Universitätsverlag Konstanz, Konstanz, 1990*, S. 13-18

[LaMa04] Mounia Lalmas, Saadina Malik:
INEX 2004 Retrieval Task and Result Submission Specification
In: *INEX 2004 Workshop Pre-Proceedings, Dagstuhl, 6. - 8. Dezember, 2004*, S. 237–240

[Leht04] Miro Lehtonen:
EXTIRP 2004: Towards heterogeneity
In: *INEX 2004 Workshop Pre-Proceedings, Dagstuhl, 6. - 8. Dezember, 2004*, S. 183–187

[LLD+02] Robert W.P. Luk, H. V. Leong, Tharam S. Dillon, Alvin T.S. Chan, W. Bruce Croft, James Allan:
A Survey in Indexing and Searching XML Documents
In: *Journal of the American Society for Information Science and Technology, Band 53, Nummer 6, John Wiley & Sons, Inc., New York (NY), U.S.A., 2002*, S. 415–437

[MaMa03] Yosi Mass, Matan Mandelbrod:
Retrieving the most relevant XML Components
In: *INEX 2003 Workshop Proceedings, Dagstuhl, 15. - 17. Dezember,, 2003*, S. 53–58

[MaMa04] Yosi Mass, Matan Mandelbrod:
Relevance Feedback for XML Retrieval
In: *INEX 2004 Workshop Pre-Proceedings, Dagstuhl, 6. - 8. Dezember, 2004*, S. 154–157

[MaMi04] Frank Manola, Eric Miller:
RDF Primer
World Wide Web Consortium, Februar 2004
(Verfügbar unter: http://w3c.org/TR/rdf-primer)

[MMA+02] Yosi Mass, Matan Mandelbrod, Einat Amitay, David Carmel, Yoelle Maarek, Aya Soffer:
JuruXML – an XML retrieval system at INEX'02
In: *Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval (INEX), Dagstuhl, 11. - 8. Dezember, 2002*, S. 73–80

[PeTV03] Jovan Pehcevski, James Thom, Anne-Marie Vercoustre:
RMIT INEX experiments: XML Retrieval using Lucy/eXist
In: *INEX 2003 Workshop Proceedings, Dagstuhl, 15. - 17. Dezember, 2003*, S. 134–141

[Port80] M. F. Porter:
An algorithm for suffix stripping
In: *Program, Ausgabe 14, Nummer 3, Juli 1980*, S. 130–137

[Rijs79] C. J. van Rijsbergen:
Information Retrieval, 2nd Edition
Buttersworth, London, England, 1979

[RoSJ76] S. Robertson, K. Sparck Jones:
Relevance Weighting of Search Terms
In: *Journal of the American Society for Information Science 27, American Society for Information Science and Technology, 1976*, S. 129–146

[ScTW04] Ralf Schenkel, Anja Theobald, Gerhard Weikum:
XXL @ INEX 2003
In: *INEX 2003 Workshop Proceedings, Dagstuhl, 15. - 17. Dezember, 2003*, S. 59–66

[ThWe02] Anja Theobald, Gerhard Weikum:
The Index-based XXL Search Engine for Querying XML Data with Relevance Ranking
In: *Proceedings of the 8th International Conference on Extending Database Technology (EDBT), Springer-Verlag, London, England, März, 2002*, S. 477–495

[TLDP05] o.V.:
The Linux Documentation Project
Linux Documentation Project, 2005
(Verfügbar unter: http://www.tldp.org)

[VrKL04] Arjen P. de Vries, Gabriella Kazai, Mounia Lalmas:
Evaluation Metrics 2004
In: *INEX 2004 Workshop Pre-Proceedings, Dagstuhl, 6. - 8. Dezember, 2004*, S. 249–250

[WaMu99] Norman Walsh, Leonard Muellner:
DocBook: The Definitive Guide
O'Reilly & Associates, Sebastopol (CA), U.S.A., 1999

[WoGe04] Alan Woodley, Shlomo Geva:
NLPX at INEX 2004
In: *INEX 2004 Workshop Pre-Proceedings, Dagstuhl, 6. - 8. Dezember, 2004*, S. 188–195

[YBP+04] François Yergeau, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler:
Extensible Markup Language (XML) 1.0 (Third Edition)
World Wide Web Consortium, Februar 2004
(Verfügbar unter: http://w3c.org/TR/2004/REC-xml-20040204)