Studienarbeit

Konzeption einer Suchmaschine für XML-Dokumente

Mathias Zarick

Betreuer: Dr. Meike Klettke

Version vom 7. September 2001

UNIVERSITÄT ROSTOCK
FACHBEREICH INFORMATIK
LEHRSTUHL DBIS
ALBERT-EINSTEIN-STRASSE 21
18059 ROSTOCK

Zusammenfassung - Abstract

Das Dokumentenaustauschformat XML gewinnt immer mehr an Bedeutung. Die Anzahl an XML-basierten Anwendungen steigt immer weiter. Diese Studienarbeit untersucht die Möglichkeiten der Suchbarkeit auf dieser neuen Domäne.

Dafür wurde eine Suchmaschine für XML-Dokumente konzipiert. Grundlage dafür waren die bekannten Techniken aus herkömmlichen Suchmaschinen auf Webinhalten.

Außerdem werden auf dem Markt vorhandene kommerzielle Lösungen vorgestellt, die eine solche Funktionalität bereitstellen. Untersucht wurden GoXML Search von XML Global Technologies, XYZFind von XYZFind und der TEXTML Server von IXIASOFT.

Aufbauend auf dem entstandenen Konzept wurde ein Prototyp implementiert. Diese prototypische Suchmaschine erlaubt ein “simples” Suchen auf XML-Inhalt und liefert damit einen Ansatzpunkt für weitergehende theoretische und praktische Betrachtungen dieser Thematik.

Inhalt

1 Einleitung

Das Dokumentenaustauschformat XML (E*X*tensible *M*arkup *L*anguage) ist immer mehr auf dem Vormarsch. Es gewinnt im World Wide Web immer größere Bedeutung. Mittlerweile sind viele Unternehmen bestrebt, Aktivitäten auf dem Gebiet XML zu tätigen, um den Anschluss auf diesem Gebiet nicht zu verlieren. Die aktuelle Spezifikation [1] für XML trägt noch immer die Versionsnummer 1.0, für sie ist aber seit dem 6. Oktober 2000 eine "Second Edition" verfügbar.

XML ist ein Markup-Format, welches sich als Format für das Austauschen von Daten zwischen verschiedenen Anwendungen eignet. Sein Einsatz ist aber auch für andere Verwendungen zweckvoll, zum Beispiel für die Bereitstellung von Daten in Informationssystemen, für die Archivierung von Daten und für vieles anderes. XML hat dabei den Vorteil, dass die enthaltenen Markup-Anteile nicht das Layout der Informationen - wie zum Beispiel bei HTML - repräsentieren, sondern die Bedeutung der Informationen.

In dieser Studienarbeit ist eine Konzeption einer Suchmaschine auf XML-Daten entwickelt worden. Bei einer solchen Suchmaschine sind für eine Auswertung sowohl die Struktur der Dokumente, also das Markup, als auch die Informationen selbst interessant.

Die Betrachtungen zu dieser Suchmaschine basieren auf den Funktionsweisen herkömmlicher Suchmaschinen. Diese suchen in der Regel auf HTML-Daten. Deren grundlegende Konzepte werden hier beleuchtet und für die neue Domäne XML erweitert.

Weiterhin werden auf dem Markt erhältliche XML-Suchmaschinen vorgestellt.

2 Vorbetrachtungen

2.1 Herkömmliche Suchmaschinen

Suchmaschinen, wie wir sie aus dem World Wide Web kennen, begegnen uns mittlerweile jeden Tag. Zu ihren berühmtesten gehören solche wie Google, Yahoo oder Altavista. An dieser Stelle soll die Funktionsweise solcher Suchmaschinen in ihren Grundzügen erklärt werden. Sie suchen in der Regel auf HTML-Daten. In der Abbildung 2.1 ist die Funktionsweise übersichtlich grafisch dargestellt.

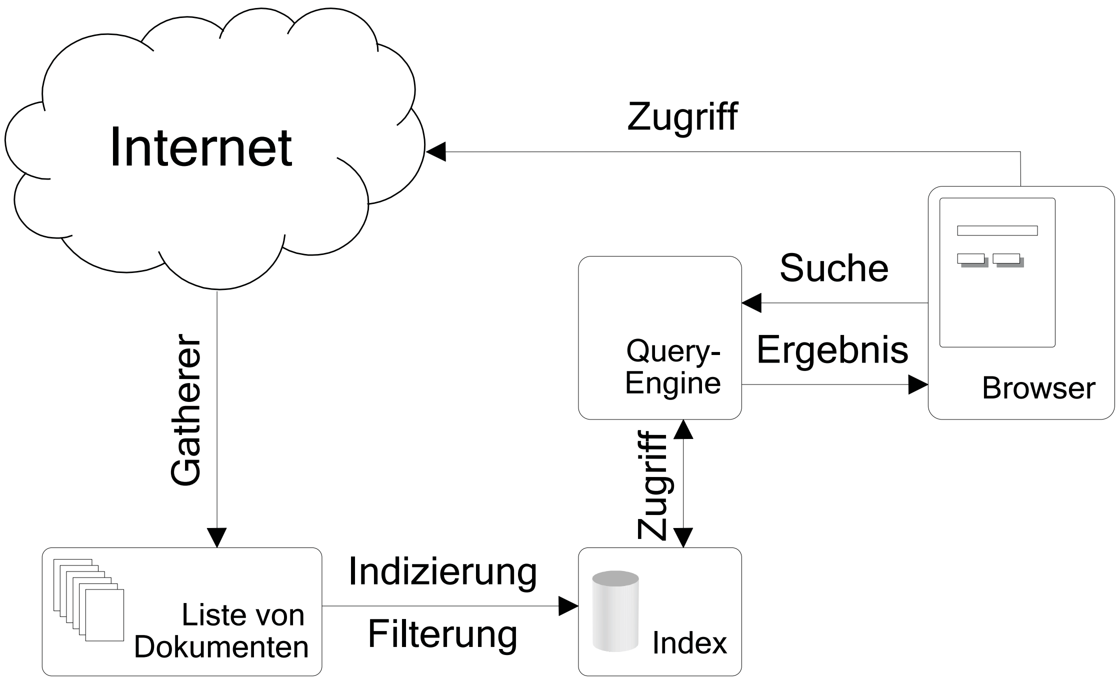


Abbildung 2.1: Grundlegende Funktionsweise einer Suchmaschine im Überblick

Die hier beschriebenen Mechanismen innerhalb solcher Suchsysteme stellen die Grundtechniken dar. Viele heute bestehende Systeme sind immer intelligenter geworden. Sie nutzen dabei beispielsweise immer mehr die Struktur von HTML-Dateien aus. Diese Idee ist mit der für diese Studienarbeit vergleichbar.

Die URLs der zu untersuchenden Dokumenten aus dem Internet werden über Gathering-

Komponenten erkundet. Die Gatherer gehen in der Regel wie folgt vor: Es gibt eine gewisse Grundmenge an bekannten Dokumenten, die den Systemen gegeben werden. Sie dienen als Einstiegsmenge für das Erstellen einer Liste zu indizierender Dokumente. Die bekannten Dokumente werden auf Links untersucht. Enthalten sie Links auf andere Dokumente, so werden diese mit in die zu untersuchende Menge aufgenommen. Dokumente die nicht verfügbar sind ("toter Link") werden aus der Menge wieder entfernt. Dieser Prozess vollzieht sich dabei iterativ solange bis keine neuen Dokumente mehr hinzugewonnen werden, also bis die Kardinalität der Menge konstant ist. Sie nutzen mehrere Einstiegspunkte und bekommen so eine sehr große Menge von Dokumenten. Diese Menge bildet den Suchraum oder die Suchdomäne. Ein Nutzer der Suchmaschine soll nun nach Dokumenten aus dieser Suchdomäne suchen können. Dazu ist es vorher notwendig, alle Dokumente aus der Suchdomäne entsprechend den Anforderungen zu indizieren.

Vor dem Indizieren werden die Dokumente in der Regel durch einen Filter geführt. Diese filtern aus den Dokumenten ihren eigentlichen Text-Inhalt, denn nur dieser spielt bei den einfachen Suchmaschinen eine Rolle. Es werden demnach meist HTML-zu-TEXT-Filter verwandt.

Das Indizieren bildet die Grundlage für eine spätere Suche. Damit nach dem Absenden der Suchanfrage nicht erneut jedes Dokument aus der Suchdomäne durchsucht werden muss, wird im vorhinein ein Index erstellt. Dieser Index enthält sogenannte Deskriptoren, Informationen über das Vorkommen einzelner atomarer Begriffe aus den Dokumenttexten. Dieses Verfahren ist statistisch und wortbasiert. Es gibt mehrere Techniken für das Aufnehmen eines Wortes in den Index. Die am häufigsten angewandte Methode ist das Stichwortverfahren, bei welchem alle Worte aus dem Text in den Index aufgenommen werden. Diesen nennt man dann auch Glossar.

Zusätzlich kann es eine sogenannte Stoppwortliste geben. Das ist eine Negativliste häufig vorkommender Wörter, wie z.B. *und* oder *der*, die wahrscheinlich nicht charakteristisch für die Texte sind. Die Wörter in der Negativliste werden nicht zu Deskriptoren. Zusätzlich werden aus der Deskriptorenliste oftmals die am häufigsten vorkommenden und die am seltensten vorkommenden Wörter gestrichen, weil eine sprachlich statistische Regel vorgibt, dass solche Wörter ebenfalls nicht charakteristisch für den Text sind und den Index nur stark vergrößern würden.

Manche Anwendungen mit speziellen Anforderungen machen eventuell eine Organisation des Indexes in einer anderen Form notwendig. Zum Beispiel ist es für eine linguistische Suche in den Texten notwendig, die Deskriptoren einer morphologischen Reduktion zu unterziehen. Dabei wird von vorkommenden Wörtern die Grundform ermittelt und im Index festgehalten. Jedes Wort fällt in eine Flexionsklasse, die durch ihre Grundform bestimmt ist. Dadurch ist eine erweiterte Suche möglich, die auch auf andere Flexionsformen eines Wortes abzielt.

Je nach Anforderung an die Anwendung wird die Granularität des Indexes gewählt. Diese gibt an, in welcher Genauigkeit die Informationen zu den einzelnen Deskriptoren im Index abgespeichert werden. Manche Anwendungen speichern einfach nur das Wort und das dazugehörige Dokument. Andere legen im Gegensatz dazu oftmals auch die Lokalität des Wortes, also beispielsweise Zeilen- oder Wortnummer, im Dokument ab.

Das klassische Verfahren für den Aufbau eines Indexes ist die invertierte Liste. Sie beinhaltet alle Deskriptoren mit ihren zugehörigen Informationen in einer nach Deskriptoren sortierten Liste. Aufgrund der Sortierung der Liste ist es einer Query-Engine möglich, den Index einfach und effizient zu durchsuchen.

Die Query-Engine bekommt dabei Anfragen von einem Benutzer, die dieser in der Regel über einen Browser absetzt. Sie wertet diese Anfragen mit Hilfe des Indexes aus und liefert dem Benutzer die URLs der passenden Dokumente in seinen Browser zurück.

Für die Aktualität der Ergebnisse ist es erforderlich, den Index auf einem neuesten Stand zu halten. Dafür ist es notwendig, die Suchdomäne regelmäßig neu durch das Gathering zu ermitteln und deren Dokumente neu zu indizieren.

2.2 XML-Umfeld

Für Suchmaschinen, die speziell für XML-Dokumente konzipiert sind, enstehen neue Chancen und erweiterte Möglichkeiten. Durch die spezielle Struktur von XML-Dokumenten wird es möglich sein, präzisere Anfragen zu stellen. Wenn man jetzt die Techniken von herkömmlich bekannten Suchmaschinen auf die für XML-Dokumente überträgt, kommt es dabei zu folgendem Umfeld:

Das Gathering einer Suchdomäne wird vorerst schwer fallen. Es ist nicht so ohne weiteres wie bei HTML möglich, die Domäne immer wieder durch das Verfolgen von Links zu erweitern. Zwar gibt es die Idee der Links für XML beim W3C mit der “XML Linking Language (XLink) Version 1.0” [2] auch, aber der Status dieser Aktivität liegt erst bei “Proposed Recommendation” und wird damit bisher so gut wie nicht verwendet. Die Adressen der XML-Dokumente für die Suchdomäne müssen einem Indexierer daher zur Zeit wohl manuell übergeben werden.

Von Vorteil ist es, dass man nun im Index auch die Struktur des XML-Dokumentes berücksichtigen kann. Für jeden potentiellen Deskriptor im Index stehen Informationen über sein Auftreten im Dokument bereit. So zum Beispiel innerhalb welcher XML-Elemente er sich befindet, oder ob er selbst Character Data, Attributwert etc. ist. Es ist sinnvoll, in den Index diese zusätzlichen Informationen zu den einzelnen Deskriptoren mitaufzunehmen. Innerhalb von XML-Suchmaschinen sind dadurch auch komplexere Anfragen denkbar. So könnte man Anfragen formulieren, die auf die Struktur der Dokumente abzielen. Ein Beispiel für eine solche Anfrage wäre die folgende, welche ich hier in einer natürlichen Sprache formuliere: `'Finde alle Dokumente mit dem Wort` *`Anton`* `innerhalb des Elementes` *`Name`*`!'`

3 Existierende Lösungen

Es gibt auf dem Markt viele kommerzielle XML-Lösungen, die für die Erstellung einer Suchmaschine auf XML-Daten behilflich sein können. Ich habe im Rahmen der Studienarbeit drei Produkte von verschiedenen Herstellern untersucht. Dafür standen mir die Webseiten der Hersteller, Artikel und White-Papers zur Verfügung. Dabei konnte ich im allgemeinen jedoch nur von außen auf die Systeme blicken. Die wesentlichen Interna blieben mir verborgen bzw. ließen sich nur vermuten. Ich kann daher nur Aussagen treffen, die in der Regel aus den Prospekten der Hersteller erwachsen sind oder durch die Funktionalität der Benutzerschnittstelle ersichtlich sind.

3.1 GoXML Search

GoXML Search ist eine XML-Plattform des Unternehmens XML Global Technologies[1]. Dessen Ziel ist es, Anwendungsprogrammierern eine XML-Datenintegrationsplattform und eine auf XML basierende Plattform zur Integration von Anwendung zu Anwendung bereitzustellen. Zu ihren Produkten gehört die Go XML Foundation. Zu seinen Komponenten wiederum gehört Go XML Search.

Go XML Search stellt eine kontextbasierte XML-Suchmaschine dar, von der der Hersteller präzise Ergebnisse verspricht. Im Kern der Suchmaschine steht eine serverseitig laufende Search-Engine, die die XML-Dokumente indiziert. Außerdem bietet das System standardisierte User-Interfaces für eine Suche und das Retrieval. Die besondere Struktur von XML wird dabei natürlich ausgenutzt, um komfortablere Suchmöglichkeiten zu bieten.

Die Abbildung 3.1 zeigt den Aufbau von Go XML Search im Überblick.

Wie eine Indizierung geschieht, wie ein erstellter Index genau aussieht sowie wie sich die Anfragemöglichkeiten gestalten lassen wurde mir leider aus keiner der Quellen deutlich.

XMLGlobal bietet auf ihren Webseiten das Herunterladen einer Testversion ihres Produktes an. Man kann dort desweiteren Online-Video-Konferenzen mit Demonstrationen des Produktes und Team-Meetings beiwohnen. Diese finden unregelmäßig statt, sind aber in einem Kalender registriert. Über diesen Kalender hat man die Möglichkeit Zeitpunkte und Details über bevorstehende Meetings und Demonstrationen zu erfahren.

[1] http://www.xmlglobal.com

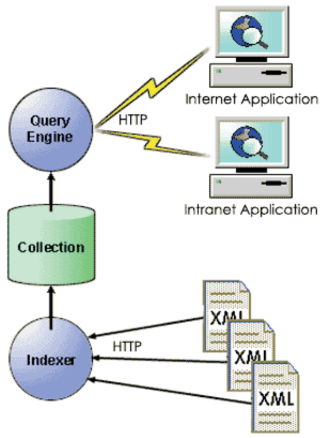


Abbildung 3.1: Go XML Search - Kernfunktionalität

3.2 XYZFind

XYZFind ist ein Produkt der XYZFind Corporation[2]. Es stellt eine schemaunabhängige XML-Datenbank dar. Sie ist in der Lage, XML-Dokumente zu speichern. XYZFind bietet eine sehr mächtige strukturierte Anfragemöglichkeit, wie man sie auch von einer relationalen Datenbank erwartet.

XYZFind kann beliebige, wohlgeformte XML-Dokumente “on the fly” speichern. Dabei parst es die Dokumente, erschließt ihre Struktur und zerlegt sie automatisch in einzelne “universelle” Datenrepräsentationen. Dabei werden alle Verbindungen zwischen den Elementen, Attributen, Character Data, Whitespace, Namespaces, Kommentaren und Processing Instructions der Dokumente festgehalten.

In dem Moment, wo XYZFind die Dokumente abspeichert und zerlegt, baut es gleichzeitig ein umfassendes Repository auf. Dieses entspricht einem Index. Jeder Aspekt von jedem Dokument wird indiziert: Text Inhalt (Character Data), Zahlen, Element- und Attributnamen, Kommentare, Processing Instructions und die Dokumentstrukturen. Alle Dokumente, ungeachtet ihrer Struktur, teilen sich ein Repository.

Das Indizierverfahren ist sehr rechenintensiv, aber dafür ist eine Anfrage sehr performant und es ist nicht notwendig, die Datenbank ständig zu “tunen”, wenn sich der Datenbestand entwickelt. XYZFind unterstützt im Gegensatz zu anderen vergleichbaren Systemen die Rückgabe von Dokumentfragmenten als Ergebnis von Anfragen. Die meisten anderen liefern nur Links auf ganze Dokumente.

XYZQL ist die Anfragesprache für die XYZFind-Datenbank. Sie selbst basiert auf XML. Sie erlaubt Einfüge-, Update-, Lösch-, Such- und Anfrageoperationen. Außerdem unterstützt sie boolesche Operatoren und Wildcard-Operationen, sowie eine Bereichssuche in einem numerischen oder alphanumerischen Bereich. Wie in traditionellen Datenbanken kann der Benutzer genau kontrollieren, welche Art von Informationen zurückgegeben werden: ganze Dokumen-

[2] http://www.xyzfind.com

te, Teile von Dokumenten, Elemente oder Attribute aus mehreren Dokumenten oder einfach nur eine Liste von passenden Dokumenten. Alle Ergebnisse werden als wohlgeformtes XML zurückgegeben. Dadurch ist es einfach, sie wieder in XML-Anwendungen zu integrieren.

Folgendes XYZQL-Beispiel für eine Anfrage liefert alle Dokumente, die die Schlüsselwörter "white" und "ford" in beliebigen Character-Data-Inhalten oder Attributwerten enthalten:

```
<xyz:input xmlns:xyz="http://xyzfind.com/schemas/xyzql/1.0">
  <xyz:query>
    <xyz:any>white ford</xyz:any>
  </xyz:query>
</xyz:input>
```

Die Anfrage liefert daraufhin ein Ergebnis zurück, das folgendermaßen strukturiert ist:

```
<xyz:output>
  <xyz:query>
    <xyz:any>white ford</xyz:any>
  </xyz:query>
  <xyz:results>
    <xyz:document name="MeinDokument.xml">
      <!-- Dokumentinhalt -->
    </xyz:document>
      <!-- Weitere Dokumente... -->
  </xyz:results>
</xyz:output>
```

Jedes Anfrageergebnis enthält als einen Teil die Anfrage, die dafür abgesetzt wurde. Ein anderer Teil sind die eigentlichen Resultate.

Der Operator `<xyz:any>` steht für einen Keyword-Search. Das heißt, die angegebenen Schlüsselwörter müssen in Character-Data-Inhalten oder Attributwerten des Dokumentes auftauchen, damit dieses sich für ein Suchergebnis qualifiziert.

Ein Auszug weiterer Anfrageoperatoren innerhalb von `<xyz:query>` ist in der Tabelle 3.1 auf Seite 15 angegeben. Einen kompletten Überblick über alle XYZQL-Statements bekommt man in dem Benutzerhandbuch zum Produkt [3].

Auf die Datenbank kann man über ein Web-Interface oder direkt über eine Java-API zugreifen.

Auch XYZFind bietet auf ihren Webseiten eine Online-Demo sowie das Herunterladen einer Testversion ihres Produktes an. Es ist sehr lohnenswert, die Online-Demo auszuprobieren. Man bekommt, nachdem man sich bei XYZFind registriert hat, seine persönliche XYZ-Datenbank gestellt. Diese besteht dann für 7 Tage oder länger, wenn man eine Verlängerung beantragt. An diese Datenbank kann man dann XYZQL-Statements absetzen, d.h. man hat volle Funktionalität gegeben, die man hier ausgezeichnet testen kann. Das Ganze funktioniert über ein Web-Interface. Man setzt XYZQL-Statements ab und bekommt das Resultat sofort in einem Popup-Browser-Fenster präsentiert.

3.3 TEXTML Server

Der TEXTML Server ist ein Produkt von IXIASOFT[3]. Auch diese Plattform stellt eine XML-Datenbank dar. Sie speichert und indiziert XML-Inhalte und unterstützt daraufhin ein Retrieval.

Der TEXTML Server beherbergt ein Repository und vom Benutzer konfigurierte Indizes. Das Repository beinhaltet die XML-Dokumente, kann aber auch jede andere Binärdatei abspeichern. Das Leistungsverhalten für den Zugriff auf die Dokumente ist dabei unabhängig vom Dateisystem des Betriebssystems.

Der TEXTML Server prüft beim Einfügen von XML-Dokumenten nicht auf ihre Gültigkeit in Bezug auf eventuell assoziierte DTD's [1] oder XML-Schemas [5, 6, 7]. Er überprüft lediglich, ob das Dokument wohlgeformt ist.

Auch der TEXTML Server parst die XML-Dokumente und erstellt Indizes, der die Informationen des XML-Markups aufnimmt. Beim TEXTML Server ist es möglich, Einfluss auf die Beschaffenheit der Indizes zu nehmen. Es ist zum Beispiel möglich zu bestimmen, bis zu welcher Verschachtelungstiefe die XML-Elemente berücksichtigt werden sollen. Desweiteren kann man Bedingungen für das Aufnehmen eines XML-Objektes in einen Index angeben. Dadurch hat man die Möglichkeit, die Struktur der Indizes auf einen bestimmten Anwendungsfall "zuzuschneiden". Die Struktur und Art der Indizes kann jederzeit verändert werden, zum Beispiel wenn sich die Anforderungen an die Anwendung verändern.

Es werden Volltextindizes, sowie Indizes für Datum, Strings und numerische Werte unterstützt. Zusätzlich werden auch Listenindizes unterstützt, wodurch die Suche nach Phrasen ermöglicht wird.

In Systemdokumenten, welche auch im Repository abgelegt werden, wird festgelegt, wie die Indizes aufgebaut sein sollen. Diese Systemdokumente sind selbst wieder XML-Dokumente. Folgendes zeigt ein Beispiel für eine solche Indexdefinition:

```
<?xml version="1.0"?>
<indexdefinition VERSION="1.0">
  <indexes>
    <index NAME="MyFullTextIndex" TYPE="FullText"
           UNINDEXABLEWORDS="FullTextStopWordList">
      <admindescription>My Full Text Index</admindescription>
      <elements>
             ⋮
      </elements>
    </index>
  </indexes>
</indexdefinition>
```

Wie die Beschreibung der einzelnen Indexelemente (*<elements>*) per XML genau aussieht wurde mir leider nicht ersichtlich.

Genauso wie die Indizes selbst lassen sich auch die Zeitpunkte der Indizierung konfigurieren. Der TEXTML Server hat einen Trigger Agent, der sich frei konfigurieren lässt. Man kann eine Indizierung zu bestimmten Ereignissen starten lassen. Folgende Ereignisse werden zum

[3] http://www.ixiasoft.com

Beispiel unterstützt: das Einfügen einer bestimmten Anzahl von Dokumenten, das Einfügen einer bestimmten Menge von Informationen (KB), bestimmte fest Zeitpunkte, das Verstreichen einer bestimmten Zeit seit dem letzten Indizierungsprozess. Man kann diese Prozesse aber auch manuell starten.

Der TEXTML Server unterstützt Anfragen auf Volltext, Datumsinhalte und Zeichenketten. Für die Suche kann man Boolesche Operatoren benutzen, um seine Anfrage zu kombinieren. Man kann weiterhin spezifizieren, auf welche XML-Objekte man die Anfrage ausüben möchte. Die Resultate lassen sich sortiert zurückgeben.

Der TEXTML Server kontrolliert seine Performance durch eine Load Balancing Komponente. Sie bewertet alle auszuführenden Transaktionen. Entsprechend ihrer Art und Komplexität gibt sie ihnen ein Gewicht, das den Ressourcenbedarf anzeigt, und eine Priorität. Sie verwaltet weiterhin eine eventuell auftretende Transaktionswarteschlange und kontrolliert den korrekten Ablauf von parallel verlaufenden Transaktionen.

Ein Überblick über die Struktur des TEXTML Servers ist in Abbildung 3.2 dargestellt.

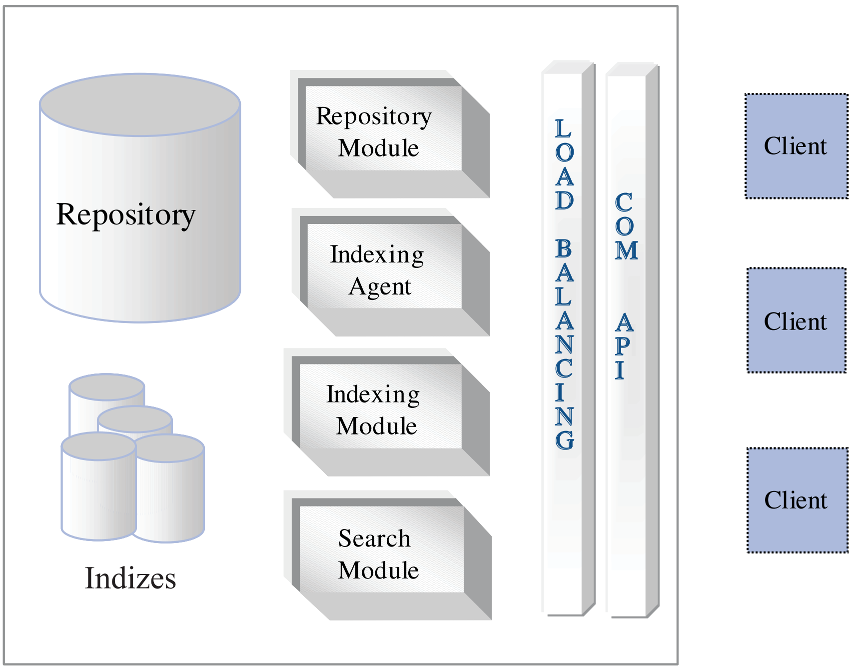


Abbildung 3.2: Aufbau des TEXTML Servers

IXIASOFT bietet auf ihren Webseiten das Herunterladen einer Testversion ihres Produktes an.

3.4 Fazit

Die Grundeigenschaften und -verfahren der untersuchten Systeme ähneln sich allesamt. Sie sind alle drei in der Lage, XML-Dokumente zu parsen und Indizes aufzubauen, welche die besondere Struktur von XML ausnutzen. Ihre Query-Engines bieten verschiedenste üppige Anfragemöglichkeiten an die erstellten Indizes. Alle Systeme versorgen ihre Nutzer mit Programmierschnittstellen (Java-APIs). Die Anwendungen, die auf den Plattformen aufsetzen, werden im Allgemeinen über einen Browser bedient.

Die Systeme unterscheiden sich hauptsächlich in der Gestaltung der Indizes, und dadurch natürlich auch in dem Angebot der Anfragemöglichkeiten. Der TEXTML Server von IXIASOFT hat den Vorteil, dass sich die Indizes genau konfigurieren lassen. XYZFind bietet wiederum eine sehr flexible Anfragemöglichkeit mittels einer intelligenten Anfragesprache XYZQL.

Weiterhin gibt es das Produkt “Allora” von Hit Software[4] und die “Tamino XML Database” der Software AG[5]. Beide genannten Unternehmen bieten mit ihren Produkten ebenfalls XML-Plattformen an, die für das Suchen in XML-Dokumenten geeignet sind. Ihre Funktionalität ist vergleichbar mit der der hier beschriebenen Systeme. “Allora” konzentriert sich jedoch mehr auf eine Speicherung der aus den XML-Dokumenten erworbenen Informationen in relationalen Datenbank-Management-Systemen. Dafür geschieht ein vom Nutzer konfiguriertes Mapping der XML-Daten auf Relationen. Weiter möchte ich aber nicht auf diese Produkte eingehen.

[4] http://www.hitsw.com
[5] http://www.softwareag.com/tamino

Anfragemethode	XYZQL-Beispiel	Erklärung

Finden von Wörtern innerhalb von Character-Data	`<xyz:query>`
	`  <author>`
	`    <name> Anton </name>`
	`  </author>`
	`</xyz:query>`
		Diese Anfrage sucht nach denjenigen Dokumenten aus dem Index, in denen das Wort "Anton" innerhalb eines <*name*>-Elementes vorkommt, das wiederum Kindelement eines Wurzelelementes <*author*> sein muss.

Finden von Wörtern innerhalb von Attributwerten	`<xyz:query>`
	`  <author>`
	`    <xyz:attribute>`
	`      <name> Anton </name>`
	`    </xyz:attribute>`
	`  </author>`
	`</xyz:query>`
		Diese Anfrage sucht nach denjenigen Dokumenten aus dem Index, in denen das Wort "Anton" innerhalb eines Attributwertes für das Attribut *name* eines <*author*>-Elementes vorkommt, das außerdem Wurzelelement sein muss.

<xyz:or>-Operator, |-Operator	`<xyz:query>`
	`  <author>`
	`    <xyz:or>`
	`      <name> Anton | Paul </name>`
	`      <company> xyzfind </company>`
	`    </xyz:or>`
	`  </author>`
	`</xyz:query>`
		Diese Anfrage sucht nach denjenigen Dokumenten aus dem Index, in denen das Wort "Anton" oder "Paul" innerhalb eines <*name*>-Elementes vorkommt, das wiederum Kindelement eines Wurzelelementes <*author*> sein muss. Zusätzlich qualifizieren sich auch alle diejenigen Dokumente, die unterhalb vom Wurzelelement <*author*> ein <*company*>-Element haben, in dessen Character Data das Wort "xyzfind" vorkommt.

<xyz:and>-Operator, Zwang zum Vorkommen mehrerer Wörter innerhalb von Character-Data oder Attributwerten	`<xyz:query>`
	`  <author>`
	`    <xyz:and>`
	`      <name> Anton Meier </name>`
	`      <company> xyzfind </company>`
	`    </xyz:and>`
	`  </author>`
	`</xyz:query>`
		Wie zuvor, nur mit einer AND-Logik anstatt von OR. "Anton" und "Meier" müssen innerhalb des <*name*>-Elementes vorkommen. Die Reihenfolge und andere Worte darin spielen dabei keine Rolle. Außerdem muss im <*company*>-Element das Wort "xyzfind" vorkommen.

<xyz:one>-Operator, Vergleichsoperatoren	`<xyz:query>`
	`  <author>`
	`    <xyz:one>`
	`      <year> >1995 <=1998 </year>`
	`    </xyz:one>`
	`  </author>`
	`</xyz:query>`
		<*xyz:one*> ist ein Wildcard für ein beliebiges Element. Innerhalb des <*year*>-Elementes muss eine Zahl größer 1995 und kleiner oder gleich 1998 vorkommen.

Tabelle 3.1: Auszug der Anfragemethoden von XYZFind

4 Konzeption

In diesem Kapitel wird die Struktur und die Funktionsweise für eine Suchmaschine auf XML-Daten konzipiert. Diese Funktionalität wurde im Rahmen dieser Studienarbeit auch prototypisch umgesetzt. Diese Umsetzung ist im nächsten Kapitel *Der Prototyp* ab Seite 25 beschrieben.

4.1 Aufbau der XML-Suchmaschine

Der Aufbau meiner konzipierten XML-Suchmaschine ist in Abbildung 4.1 dargestellt.

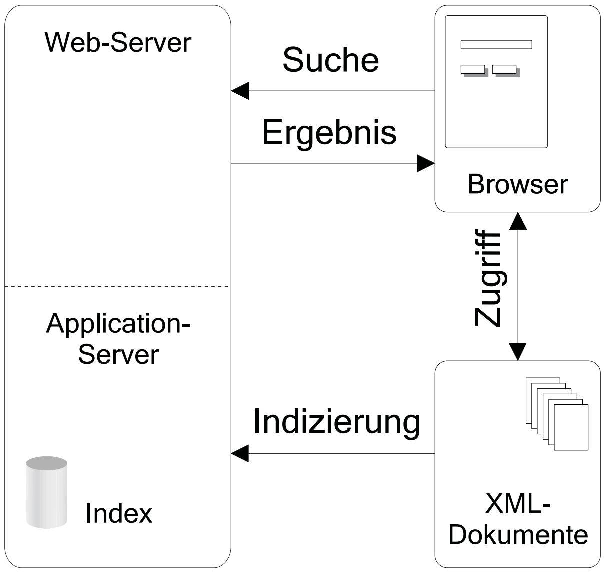


Abbildung 4.1: Aufbau der XML-Suchmaschine

Auf den ersten Blick sind viele Gemeinsamkeiten mit dem Aufbau von herkömmlichen Suchmaschinen erkennbar.

Die Suchdomäne für die Suchmaschine ist jedoch konstant und wird der Anwendung manuell übermittelt. Ein Gathering von XML-Dokumenten aus dem Internet wäre mittels der Technologie XLink (siehe auch Kapitel 2.2) denkbar. XLink wird im Moment aber kaum benutzt.

Die XML-Dokumente aus der Suchdomäne werden indiziert. Dieser Index befindet sich dann auf dem Application-Server und stellt die Voraussetzung für das Suchen nach bestimmten Dokumenten dar. Die Suchanfrage wird vom Benutzer über ein HTML-Formular an einen Webserver abgesendet. Der Webserver wiederum startet einen Suchprozess auf dem Application-Server. Dabei nutzt er für die Suche den bei sich gelagerten Index. Die Suchergebnisse werden dem Benutzer vom Webserver im Browser präsentiert. Von dort aus kann er dann auf die gefundenen Dokumente direkt zugreifen.

4.2 Indizierung

Folgende Informationen nimmt der Index auf:

Das indizierte Objekt *object* ist die Zeichenkette, die in den XML-Dokumenten vorkommt und in den Index aufgenommen wird. Aus den XML-Dokumenten werden alle atomaren Begriffe indiziert. Das können Namen von Elementen, Namen von Attributen, Werte von Attributen oder Character Data sein. $object_i$ gibt das i-te indizierte Objekt im Index an.

Die Anzahl der Vorkommen *n* gibt an, wie oft das indizierte Objekt *object* in allen Dokumenten der Suchdomäne vorkommt. n_i gibt die Anzahl der Vorkommen vom i-ten indizierten Objekt im Index an.

Die Adresse des Dokuments *address* gibt die Adresse bzw. die URL für den Zugriff auf das XML-Dokument an, in dem das indizierte Objekt *object* vorkommt. Jedem indizierten Objekt $object_i$ werden n_i Adressen *address* zugeordnet. $address_{ij}$ gibt die j-te zugeordnete Adresse vom i-ten indizierten Objekt im Index an.

Der XML-Pfad *XMLPath* ist eine Zeichenkette, die die Einbindung des indizierten Objektes *object* in die Struktur des XML-Dokumentes widerspiegelt. Diese Zeichenkette ist ähnlich aufgebaut wie die Location Path Ausdrücke aus der XML Path Language (XPath) [4]. In *XMLPath* werden jedoch lediglich Elementnamen und für Attributwerte auch der Name des Attributes aufgenommen. Jegliche weitergehende Konzepte aus XPath finden hier keine Verwendung. Man kann aus dieser Zeichenkette ablesen, unterhalb welcher Elemente sich das indizierte Objekt *object* befindet. Bei den indizierten Objekten, die Attributwerte sind, kann man zusätzlich den Namen des Attributes aus *XMLPath* ablesen.

Beispiel: In folgendem Ausschnitt aus einem XML-Dokument soll *XMLPath* für den fettgedruckten Attributwert betrachtet werden:

```
<?xml version="1.0"?>
<ATOM>
   <NAME>Chlorine</NAME>
   <ATOMIC_WEIGHT>35.4527</ATOMIC_WEIGHT>
   <ATOMIC_NUMBER>17</ATOMIC_NUMBER>
   <OXIDATION_STATES>+/-1, 3, 5, 7</OXIDATION_STATES>
   <BOILING_POINT UNITS="Kelvin">239.18</BOILING_POINT>
⋮
</ATOM>
```

XMLPath für das *object* **`Kelvin`** ist `/ATOM/BOILING_POINT/@UNITS/`.
$XMLPath_{ij}$ gibt den j-ten zugeordneten *XMLPath* vom i-ten indizierten Objekt im Index an.

Der XML-Typ *XMLType* gibt an, von welchem Typ das indizierte Object *object* ist. Unterschieden wird zwischen Elementnamen, Attributnamen, Attributwerten und Character Data.
$XMLType \in \{Element\_Name, Attribute\_Name, Attribute\_Value, Character\_Data\}$.
$XMLType_{ij}$ gibt den j-ten zugeordneten *XMLType* vom i-ten indizierten Objekt im Index an.

Außerdem treffe ich noch folgende Vereinbarungen für Bezeichnungen:
D ist die Suchdomäne – das ist die Menge aller XML-Dokumente, die indiziert wurden.
N sei die Anzahl der Indexeinträge, also die Anzahl aller *objects*.
Das Tupel $(address, XMLPath, XMLType)$ sei $IndexInfo$.
Eine Menge von solchen Tupeln sei $IndexInfoList$.
$IndexInfoList_i$ sei jenes Tupel, das dem i-ten Indexeintrag zuordenbar ist.
Das Tupel $(object_i, n_i, IndexInfoList_i)$ sei $IndexEntry_i$.
Eine Menge von solchen Tupeln sei ein $Index$.

4.3 Logische Struktur des Indexes

Folgende Zusammenhänge bestehen:
$IndexInfo_{ij} = (address_{ij}, XMLPath_{ij}, XMLType_{ij})$
$IndexInfoList_i = \{IndexInfo_{i1}, \ldots, IndexInfo_{in_i}\}$
$IndexEntry_i = (object_i, n_i, IndexInfoList_i)$
$Index = \{IndexEntry_1, \ldots, IndexEntry_N\}$
$i \in \{1, \ldots, N\}, j \in \{1, \ldots, n_i\}$

Es gilt außerdem: $object_i < object_j$ für $i < j$. Das heißt der Index wird nach seinen indizierten Objekten alphabetisch sortiert aufgebaut.

Die logische Struktur des Indexes spiegelt die Abbildung 4.2 wider:

4.4 Anfrage

Die Anfrage an die Suchmaschine besteht aus

- einem Suchstring *term*,
- einem XML-Typ *XMLType* (siehe Kapitel 4.2, optional)
- einem XML-Pfad *XMLPath* (siehe Kapitel 4.2, optional)
- einer Anfragemethode *XMLPathQueryMethod* bezüglich der Handhabung eines eventuell angegebenen XML-Pfades (optional)

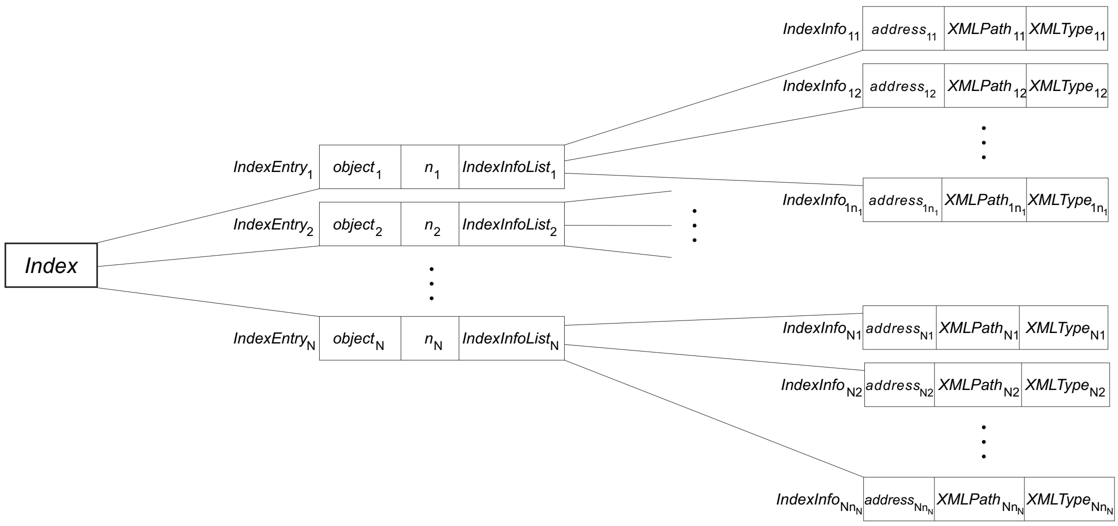


Abbildung 4.2: Logische Struktur des Indexes

term ist die Zeichenkette, nach der gesucht werden soll. Sie muss vollständig in den XML-Dokumenten auftauchen, damit diese sich für das Suchergebnis qualifizieren. Sie entspricht dem indizierten Objekt *object* aus dem Index. Eine Suche nach *term* entspricht also der Bestimmung folgender Menge: $M = \{i | term = object_i \wedge i \in \{1, \ldots, N\}\}$. Ist diese Menge M nicht leer, so gilt es im Anschluss daran noch, die dazugehörige Dokumentenmenge, also $E = \{address_{mj} | m \in M \wedge j \in 1, \ldots, n_m\}$, zu bestimmen. E ist die Ergebnismenge, sie beinhaltet alle Dokumente, die die Anfrage erfüllen. Sie wird dem Benutzer in geeigneter Form im Browser präsentiert.

Mit der Angabe eines XML-Typen mittels *XMLType* kann der Benutzer seine Auswahl weiter einschränken. Es qualifizieren sich in diesem Fall nur die Dokumente, in denen *term* als der angegebene Typ *XMLType* vorkommt. E lässt sich in diesem Falle folgendermaßen beschreiben: $E = \{address_{mj} | m \in M \wedge j \in 1, \ldots, n_m \wedge XMLType = XMLType_{mj}\}$.

Werden *XMLPath* und *XMLPathQueryMethod* angegeben, so wird bei der Suche zusätzlich der im Index abgespeicherte XML-Pfad (siehe Kapitel 4.2) berücksichtigt. Folgende Anfragemethoden bezüglich dieses Pfades gibt es:

- der angegebene *XMLPath* und die im Index abgespeicherte Zeichenkette müssen genau übereinstimmen (`EXACT_MATCH`),
- *XMLPath* muss mit den ersten Zeichen der Zeichenkette im Index übereinstimmen (`PARTIAL_MATCH_START`),
- *XMLPath* muss ein Teilstring der Zeichenkette im Index sein (`PARTIAL_MATCH_MIDDLE`),
- *XMLPath* muss mit den letzten Zeichen der Zeichenkette im Index übereinstimmen

(`PARTIAL_MATCH_END`).

Der Benutzer schränkt mit Angabe von *XMLPath* und *XMLPathQueryMethod* das Suchergebnis weiter ein. Jedoch wird dieses nur dann weiter beeinflusst, wenn beide Parameter angegeben werden.

Das Suchergebnis E sieht unter der Berücksichtigung aller 4 Anfrageparameter *term*, *XMLType*, *XMLPath* und *XMLPathQueryMethod* folgendermaßen aus:
$E = \{address_{mj} | m \in M \wedge j \in 1, \ldots, n_m \wedge XMLType = XMLType_{mj}$
$\wedge (XMLPath, XMLPath_{mj}, XMLPathQueryMethod) \in r\}$

r sei dabei eine dreistellige Relation, die wie folgt erklärt ist:
$r \subseteq String \times String \times \{EXACT\_MATCH, PARTIAL\_MATCH\_START,$
$PARTIAL\_MATCH\_MIDDLE, PARTIAL\_MATCH\_END\}$

$$(s_1, s_2, m) \in r \Longleftrightarrow: \begin{cases} s_1 = s_2, & \text{falls } m = EXACT\_MATCH \\ s_2 \text{ beginnt mit } s_1, & \text{falls } m = PARTIAL\_MATCH\_START \\ s_2 \text{ enthält } s_1, & \text{falls } m = PARTIAL\_MATCH\_MIDDLE \\ s_2 \text{ endet auf } s_1, & \text{falls } m = PARTIAL\_MATCH\_START \end{cases}$$

4.5 Weiterführende Konzeption

Die Konzeption, die in diesem Kapitel bis zu diesem Punkt beschrieben wurde, habe ich in einem Prototyp vollständig implementiert. Dieser Prototyp ist im nächsten Kapitel ab Seite 25 beschrieben. Es gibt darüberhinaus zahlreiche Erweiterungsmöglichkeiten, die die Funktionalität vergrößern würden. Diese sind Inhalt dieses Abschnittes.

4.5.1 Halten des Indexes in einem Sekundärspeicher

Der Prototyp hält den gesamten Index im Hauptspeicher. Je nach der Größe der Suchdomäne und der XML-Dokumente darin kann der Index sehr schnell ein enormes Ausmaß annehmen. Für solche Fälle ist es nicht mehr sinnvoll, den gesamten Index im Hauptspeicher zu halten. Es ist dann zweckmäßig, ihn in einen Sekundärspeicher also konkret in Dateien auf Festplatten umzulagern. Der Nachteil, der dabei aber entsteht, ist eine verzögerte Antwortzeit der Anfragen aufgrund einer größeren Zugriffszeit auf die Indexinformation.

Folgendermaßen kann es realisiert werden, den Index in einer einzigen Datei auf einer Festplatte zu halten: Die logische Struktur des Indexes (siehe 4.3) bleibt die gleiche. Die einzelnen Datensätze werden jetzt aber in anderer Form in der Datei abgespeichert. Für jeden *IndexEntry* wird eine Zeile in der Datei reserviert. Diese wird dann jedesmal bei Bedarf erweitert.

Jede Zeile beginnt zweckmäßigerweise mit einer Information, wie viele Bytes diese Zeile belegt. Dafür sollte man die ersten beiden Bytes reservieren. Das heißt, eine Zeile kann dann bis zu 65535 weitere Bytes beinhalten. Als nächstes folgt ein Byte, das angibt, wie viele Bytes das darauffolgende *object* belegt. Danach steht dieses dann. Hiernach wiederum folgt ein Byte, welches die Anzahl der Vorkommen *n* angibt. Hieran schließen sich nun noch die *n* *IndexInfo*-Datensätze zu dem aktuellen *object* an. Sie stehen hier alle einfach nacheinander. Das ist möglich, weil ja durch *n* bekannt ist, wie viele es sind. Hierin werden die *address*- und *XMLPath*-Felder, welche ja beide Stringtypen sind, jeweils durch ein Byte angekündigt, in welchem ihre Länge angegeben ist. Das *XMLType*-Feld wird einfach durch ein einziges Byte repräsentiert.

Alle hier angegebenen Größen einzelner Bytefelder sind lediglich Vorschläge. Sie können natürlich bei Bedarf je nach Applikationsanforderungen vergrößert oder auch verkleinert werden.

Eine Zeile der Indexdatei ist in folgender Grafik noch einmal dargestellt:

Zeile i	Anzahl der Bytes insgesamt	Größe des *object*-Feldes	$object_i$	n_i	Größe des *address*-Feldes	$address_{i1}$	Größe des *XMLPath*-Feldes	$XMLPath_{i1}$	$XMLType_{i1}$	. . .

Abbildung 4.3: Speicherstruktur des Indexes in einer Datei

Durch das Speichern des Indexes in einer Datei ändert sich zwangsläufig auch der Zugriff auf den Index bei einer Anfrage. Die Grundidee für den Zugriff kann jedoch die gleiche bleiben. Die Indexeinträge also hier die Zeilen werden systematisch in einem schnellen Divide-and-Conquer-Algorithmus nach dem zu findenden Term durchsucht. Dabei wird die Sortiertheit des Indexes nach den *objects* ausgenutzt.

Man beginnt für die Suche mit einer mittleren Zeile in der Datei und überprüft, ob der Suchterm genau hier steht. Falls nicht, so muss er entweder vor oder nach dieser Position stehen. Man hat also die Anzahl der noch in Frage kommenden Zeilen halbiert. Diese Vorgehensweise wiederholt sich so oft bis die Suche mit oder ohne Erfolg zum Ende kommt. Die Komplexität dieses effizienten Algorithmus ist logarithmisch.

4.5.2 Komplexe Anfragen

Eine Erweiterungsmöglichkeit für die Anfragefunktionalität wäre die Unterstützung einer komplexeren Suche. So könnte der Nutzer zum Beispiel die in Kapitel 4.4 beschriebenen atomaren Anfragen beliebig mit den Operatoren *NOT, AND, OR* kombinieren.

A_1 liefere das Suchergebnis E_1, A_2 liefere das Suchergebnis E_2. $NOT\ A_1$ liefert dann $D \backslash E_1$, $A_1\ AND\ A_2$ liefert $E_1 \cap E_2$ und $A_1\ OR\ A_2$ liefert letztendlich $E_1 \cup E_2$.

4.5.3 Ausnutzen von DTDs und XML-Schemas

Um die Anfragefunktionalität noch mehr zu erweitern, könnte man auf die XML-Dokumente abzielen, die in Bezug auf eine bestimmte DTD [1] oder auf ein bestimmtes XML-Schema [5, 6, 7] valid sind. Diese Strukturdefinitionen müssten der Query-Engine dafür übergeben werden. Diese Idee lässt sich sogar noch dahingehend erweitern, dass dem Benutzer ein auf die DTD bzw. auf das Schema "zugeschnittenes" Formular für die Suche angeboten wird. Dazu wären Forschungen dahingehend notwendig, wie man es realisieren kann, aus DTDs und oder Schemas automatisch passende Eingabemasken zu erzeugen. Die Infozone-Group[1] beschäftigt sich in

[1] Die Infozone-Group (www.infozone-group) ist eine not-for-profit Organisation, die Java- und XML-basierte Komponenten für Programmierer entwickelt.

ihrem Projekt SchemoX [8] mit diesem Thema. Allerdings wird nur XML-Schema als Strukturdefinition unterstützt. Sie schätzen den Informationsgehalt von DTDs als zu gering ein, so dass diese in dem Projekt keine Beachtung finden. Eine genauere Untersuchung dieser Thematik ist aber kein Bestandteil dieser Studienarbeit.

4.5.4 Volltextfunktionalität

Eine zusätzliche Möglichkeit, das System zu erweitern, wäre eine Kombination der XML-bezogenen Komponente mit einer Komponente, die Volltextfunktionalitäten unterstützt. Dabei könnten für die XML-Dokumente zusätzlich Volltextindizes und Indizes für eine Unterstützung von Phrasensuchen erzeugt werden. Dadurch entsteht die Möglichkeit, Anfragen wie in Text-Retrieval-Systemen abzusetzen. Dazu gehören zum Beispiel eine Phrasensuche oder auch das Verwenden von Wildcard-Operatoren in den Anfragen. Systeme, die diese Funktionalität unterstützen, sind z.B. Fulcrum und die Datenbankvolltexterweiterungen "Text-Extender" für DB2 und das "Excalibur Text Search Data Blade" für Informix.

4.5.5 XPath-Funktionalität

Mittels der XML Path Language (XPath) [4] lassen sich Teile eines XML-Dokumentes adressieren. Eine solche Adressierung könnte für die Formulierung einer konkreten Anfrage auf die XML-Suchmaschine benutzt werden.

Die Anfrage würde dann zweckmäßigerweise 2 Teile enthalten. Zum einen würde eine Adresse eines Dokumentteils in XPath enthalten sein und zum anderen wäre eine Erklärung Bestandteil, was sich an so einer Position bzw. in so einem Dokumentteil befinden soll, damit sich das Dokument qualifiziert.

Um solche Funktionalität zu erreichen, muss der Index jedoch komplizierter und umfangreicher als in meinem Ansatz aus 4.3 aufgebaut werden.

4.5.6 Integration der Ordnung in den Index

Der Index aus 4.3 eignet sich hinreichend für eine Anfrage auf XML-Inhalte. Jedoch kennt der Index nicht die Reihenfolge seiner gespeicherten Informationen innerhalb der Dokumente, in denen sie vorkommen.

An dieser Stelle möchte ich ein Beispiel anbringen, welches die Situation verdeutlicht. Folgendes XML-Dokument wird indiziert:

```
<?xml version="1.0"?>
<AUTHORS>
  <AUTHOR>
    <NAME>Schulze</NAME>
    <PRENAME>Karl</PRENAME>
  </AUTHOR>
  <AUTHOR>
    <NAME>Lehmann</NAME>
    <PRENAME>Hans</PRENAME>
  </AUTHOR>
```

```
</AUTHORS>
```

Eine komplexe Anfrage (siehe 4.5.2), die auf einen Autoren *Hans Schulze* abzielt, liefert bei diesem Dokument bei der bisherigen Gestaltung des Indexes Erfolg, obwohl das so nicht sinnvoll ist.

Gesucht wird hierbei nach `Schulze` unter dem XML-Pfad `\AUTHORS\AUTHOR\NAME\` und nach `Hans` unter dem XML-Pfad `\AUTHORS\AUTHOR\PRENAME\`. Jedoch wird nicht beachtet, dass das `<AUTHOR>`-Element dasselbe sein sollte, damit es eine sinnvolle Anfrage ist.

Jedes einzelne Element im abgespeicherten Pfad müsste also zusätzlich noch eine Ordnungszahl mitaufnehmen, welche angibt, um das wievielte XML-Element es sich handelt. Die Pfadinformation im Index könnte dann mit einer diesbezüglichen Verbesserung folgendermaßen aussehen: `\AUTHORS(R)\AUTHOR(1)\NAME(1)\`

`(R)` steht für das Root-Element und die Zahlen in Klammern geben die Nummer dieses Elementes an.

Damit die Anfrage von oben nun ein sinnvolles Ergebnis liefert, muss nur noch sichergestellt werden, dass es sich in den beiden XML-Pfaden um dieselben `<AUTHOR>`-Elemente handelt. Das heißt, die Ordnungszahlen dieser Elemente müssen übereinstimmen.

4.5.7 Aufbau von XQuery- oder XQL-Systemen

Die Konzepte für die XML-Anfragesprachen, wie z.B. XQuery [9] und XQL [10], die sich noch in ihrer Geburtsstunde befinden, liefern zusätzliche Ansatzpunkte, die man in die Konzepte für XML-Suchmaschinen einbinden kann. Solche Systeme, die die Funktionalität der Anfragesprachen umsetzen, könnten auch umgekehrt XML-Suchmaschinen für einen Teil ihrer Aufgabe benutzen. Dabei könnten sie wie folgt vorgehen:

Es wird eine Anfrage in XQuery oder XQL abgesetzt. Diese soll für eine große Vielzahl von Dokumenten ausgeführt werden. Wenn die Anfragesysteme nun jedes einzelne Dokument auf die Anfrage hin betrachten, erfordert dieser Prozess einen enorm hohen Aufwand. Zweckmäßig ist es, die zu untersuchende Menge von Dokumenten vor Absenden der eigentlichen Anfrage einzuschränken, und zwar auf diejenigen, die überhaupt auf die Anfrage passen.

Alle Dokumente, die nicht wohlgeformt sind, oder alle diejenigen, die nicht valid in Bezug auf eine DTD oder ein XML-Schema sind, könnten sofort aus der Betrachtung gezogen werden. Alle die Dokumente, denen eine DTD oder ein XML-Schema zugeordnet ist und denen sie auch genügen, bieten ebenfalls einen Ansatz für das Außerachtlassen. Man kann nämlich die DTD bzw. das XML-Schema zuerst daraufhin untersuchen, ob die zugeordneten Dokumente überhaupt auf die Anfrage passen können. Ist das nicht der Fall, so sollten die Dokumente ebenfalls außer acht gelassen werden.

Schließlich bietet eine XML-Suchmaschine eine weitere Möglichkeit der Einschränkung der Dokumentenmenge, auf die die Anfrage letztendlich abgesetzt wird. Und zwar könnten sie in folgender Weise benutzt werden:

Die XML-Anfrage wird zunächst analysiert. Aufsetzend auf diese Analyse werden in einer ersten Stufe Suchanfragen an die Suchmaschine abgesetzt, die die Dokumentenmenge wiederum einschränkt. Dafür muss der Zweck der XML-Anfrage erkannt werden und die Suchanfrage an die XML-Suchmaschine nach gewissen Heuristiken korrekt aufgebaut werden. Diese Anfragen dürfen auf keinen Fall zu “scharf” werden, so dass Dokumente wegfallen würden, die auf jeden Fall wesentlich für die XML-Anfrage gewesen wären. Im zweiten Schritt wird dann schließlich die eigentliche XML-Anfrage an eine reduzierte Menge von Dokumenten abgesetzt. Diese Idee ist in der Abbildung 4.4 illustriert:

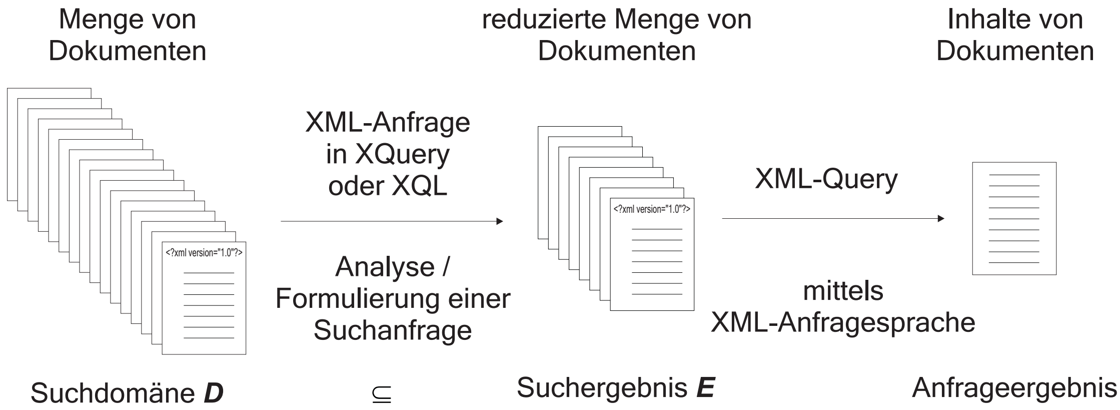


Abbildung 4.4: Nutzung der XML-Suchmaschine für XML-Anfragesprachen

Durch diese Vorgehensweisen könnten die Systeme einen erheblichen Performancegewinn erreichen.

5 Der Prototyp

5.1 Aufbau

Der Aufbau des Prototypes entspricht dem konzipierten Aufbau aus Abbildung 4.1. Dabei wurden konkret folgende Systeme benutzt. Als Webserver wurde Tomcat von Apache [11] gewählt. Tomcat ist ein Teilprojekt von Jakarta. Es gilt als offizielle Referenzimplementation für die JavaServer Pages und Java Servlet Technologien. Letztere wird für die Implementation des Prototypes genutzt, gerade deshalb bietet sich Tomcat als Webserver geradezu an.

Der Application-Server ist ebenfalls Tomcat. Er führt Java-Code in Form von einem Java-Servlet serverseitig aus. Dieses Servlet stellt das wesentliche Modul der Search-Engine dar.

Zum Test des Systems habe ich 112 XML-Dokumente genutzt. Jedes dieser Dokumente repräsentiert eines der chemischen Elemente und beinhaltet die Informationen des Periodensystems der Elemente. Für das Erstellen dieser einzelnen Dokumente habe ich [12] und den XML-Editor XML Spy[1] genutzt.

Außerdem habe ich ein XSL-Stylesheet [13] für alle Dokumente erstellt und eingebunden. Dieses Stylesheet bewirkt eine transformierte Darstellung der Dokumente in einem Browser. Jedoch unterstützen die aktuellen Browser die XSL-Stylesheets nicht bzw. mangelhaft. Lediglich der Microsoft Internet Explorer ab Version 5.5 bindet diese Stylesheets ein. Er interpretiert diese aber nur dann richtig, wenn man auf dem entsprechenden System den Microsoft XML Parser Version 3.0[2] im Replace Mode installiert.

5.2 Umfeld und Voraussetzungen

JDK Für die Implementierung wurde *Java^{TM} 2 SDK, Standard Edition v. 1.3* von Sun Microsystems genutzt.

Xerces Für die Implementierung des Prototypes habe ich die Java-Klassen des Xerces-Parsers von Apache [14] verwendet. Diese werden für das Parsen der XML-Dokumente benutzt.

CLASSPATH-Variable In der CLASSPATH-Umgebungsvariable müssen folgende Einträge vorhanden sein:

- **.** , das aktuelle Verzeichnis,

[1]http://www.xmlspy.com
[2]http://msdn.microsoft.com/xml/general/xmlparser.asp

- `xerces.jar`, die Klassenbibliothek für Xerces,
- `lib\servlet.jar`, die Klassenbibliothek für die Servletunterstützung (wird mit Tomcat mitgeliefert),
- `lib\webserver.jar`, die Klassenbibliothek für die Funktionalität von Tomcat (wird ebenfalls mit Tomcat mitgeliefert),

Webserver Damit der Webserver wie gewünscht funktioniert, muss dessen Konfigurationsdatei angepasst werden. Das ist die Datei *server.xml* aus dem Unterverzeichnis *conf* von Tomcat. Hier wird unter anderem der IP-Port für den Webserver festgelegt. Außerdem erklärt man in dieser Datei, wo sich die Dateien für die Anwendung befinden. Gemeint sind damit ein HTML-Formular und die Java-Klassen für die Anwendung.

Mit folgendem Eintrag in der Datei `conf\server.xml` wird dem Webserver diese Information vermittelt.

```
<Context path="/XMLSearch"
         docBase="webapps/XMLSearch"
         crossContext="true"
         debug="9"
         reloadable="true"
         trusted="false" >
</Context>
```

Das Attribut *path* beschreibt, über welchen Pfad die Anwendung über den Webserver von außen zugänglich sein sollen, hier */XMLSearch*. Es ist in der URL also an den Servernamen */XMLSearch* anzuhängen, um via HTTP auf die Anwendung zuzugreifen. *docbase* gibt das Verzeichnis an, in welchem sich die Dateien im Dateisystem befinden. Das ist hier relativ zum Homedirectory des Webservers geschehen. In *webapps/XMLSearch* befinden sich also die Dateien. Auf die anderen Attribute möchte ich hier nicht näher eingehen.

In dem Verzeichnis `webapps/XMLSearch/WEB-INF/classes` befinden sich die Java-Klassen der Anwendung, diese müssen in einer Verzeichnisstruktur entsprechend ihrer Packagezugehörigkeit abgespeichert werden.

In der Datei `webapps/XMLSearch/WEB-INF/web.xml` ist das Servlet-Mapping erklärt. In ihr wird beschrieben, welches Java-Servlet über welches URL-Pattern aufgerufen wird.

Nach Vornehmen all dieser Konfigurationen kann man den Webserver und damit das ganze System starten. Dazu führt man das Skript `bin/startup.sh` auf UNIX-Systemen bzw. `bin/startup.bat` auf Windows-Systemen aus. Beim Starten des Webservers wird das Servlet initialisiert. Zur Initialisierung gehört unter anderem der Aufbau des Indexes. Den Webserver kann man mit dem Skript `bin/shutdown.sh` bzw. `bin/shutdown.bat` wieder herunterfahren.

5.3 Implementierung

Der Quelltext aller hier beschriebenen Komponenten ist im Anhang nachzulesen.

Die elementare Klasse des Systems ist die Klasse `XMLQuery`. In ihr findet sich der Einstieg in die Suchmaschine. Sie realisiert die Servlet-Funktionalität.

Beim Initialisieren des Servlets mittels der Methode `init()` wird die Konfigurationsdatei `XMLSearch.properties` ausgelesen. Sie muss sich in demselben Verzeichnis wie die Klassen befinden. Über diese Datei lassen sich gewisse Parameter für das System konfigurieren. Man kann hier zum Beispiel einstellen, wo sich die XML-Dateien befinden, die die Suchdomäne bilden oder eine Validierung der XML-Dokumente gegen eine DTD oder ein XML-Schema an- bzw. ausschalten.

Weiterhin ist diese Klasse für das Logging der abgesetzten Anfragen verantwortlich. In der Konfigurationsdatei `XMLSearch.properties` gibt man an, wo diese zu erstellen ist. Im Logfile werden pro Lebenszeit eines Servlets alle Anfragen protokolliert. Für jede Anfrage werden dabei alle Parameter, die das Servlet durch den HTTP-Request erhalten hat, festgehalten. Das Logfile bietet damit eine Grundlage für eine Erweiterung des Systems um eine statistische Komponente.

Im folgenden ist ein Eintrag in das Logfile angegeben, welcher beim Absetzen einer Anfrage an die Suchmaschine mitprotokolliert wurde:

```
Date: 04.06.01 16:41
Cache-Control: no-cache
Host: localhost:8080
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Content-Length: 83
Accept-Language: de
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Connection: Keep-Alive
Referer: http://localhost:8080/XMLSearch/index.html
Method: POST
Request URI: /XMLSearch/XMLQuery
Protocol: HTTP/1.1
PathInfo: null
Remote Address: 127.0.0.1
phrase: 0
XMLPath: electronegativity
XMLType: 3
XMLPathQueryMethod: 2
```

In der Klasse ist außerdem die Funktionalität für das Parsen der XML-Dokumente implementiert. Es wird der in der Konfigurationsdatei `XMLSearch.properties` festgelegte SAX-Parser verwendet. Die auszuführenden Reaktionen auf die einzelnen SAX-Ereignisse sind ebenfalls in `XMLQuery` implementiert.

Die andere wichtige Klasse des Systems ist die Klasse `XMLIndex`. Sie realisiert die Indizierungsfunktionalität. Ihre wesentlichen Methoden sind `add` und `query`. Diese werden beide aus `XMLQuery` aufgerufen. Eine Instanz dieser Klasse stellt die Indexstruktur dar. Durch die Methode `add` wird der Index um einen Objekteintrag erweitert. Die Objektinformationen werden über die 4 Parameter von `add` übergeben. Mittels der Methode `query` werden Anfragen aus `XMLQuery` an den Index gestellt. Das Resultat dieser Methode ist eine Liste von Dokumenten, die die Anfrage erfüllt haben.

Die Aufgabe der anderen implementierten Klassen ist es, die Datentypen zu Verbünden zu bündeln bzw. benutzte Konstanten zu definieren.

Schließlich habe ich noch ein HTML-Formular `index.html` entwickelt. Es stellt den Einstieg in die Suchmaschine, also das Suchformular dar. Hier kann der Nutzer seine Anfragen absetzen. Es besitzt also einen Verweis auf das Servlet.

Für die Implementierung nutzte ich [15] und [16].

5.4 Bedienung

Der Einstiegspunkt für das XML-Suchsystem ist das Suchformular, welches dem Benutzer in einem Browser präsentiert wird. Auf dieses kann via *http* zugegriffen werden. Die URL für das Formular ist von einigen Konfigurationsparametern bestimmt. Mit folgenden Einstellungen habe ich das System im Fachbereich Informatik installiert:

- Workstation: `Pella.Informatik.Uni-Rostock.de`
- IP-Port: `8580`
- Zugriffspfad zur Anwendung: `/XMLSearch`

Daraus ergibt sich als URL für den Zugriff auf das Startdokument `http://Pella.Informatik.Uni-Rostock.de:8580/XMLSearch`.

Das Startdokument ist in Abbildung 5.1 dargestellt. Mit diesem Formular kann der Benutzer seine Anfragen an die Suchmaschine absetzen.

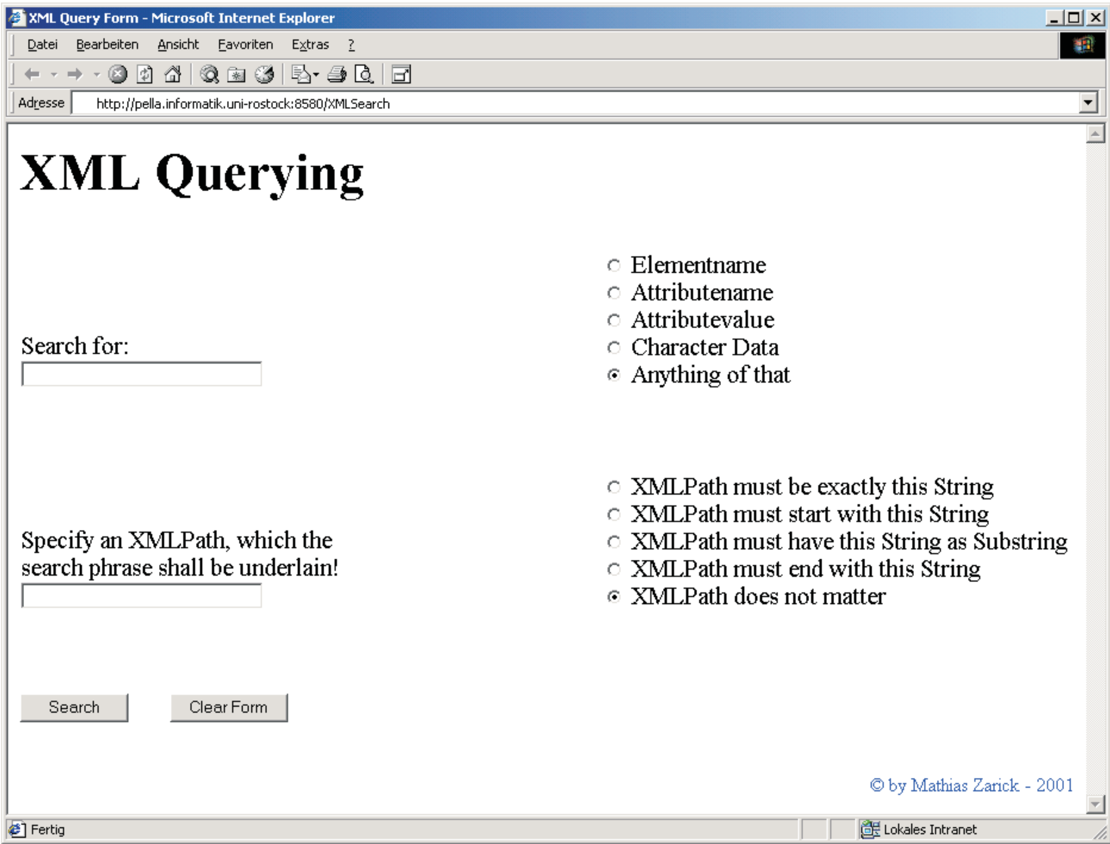


Abbildung 5.1: Das Suchformular

Das Suchformular enthält zwei Textfelder und zwei Gruppen von Radio-Buttons. In dem ersten Textfeld gibt der Benutzer den Suchstring an. Dieser muss mit einem Deskriptor im Index genau übereinstimmen, damit sich das dazugehörige Dokument für das Suchergebnis qualifiziert. Mit der daneben liegenden Radio-Button-Gruppe kann zusätzlich eingeschränkt werden, von welchem Typ dieses Objekt sein muss.

Das zweite Textfeld und die zweite Radio-Button-Gruppe ermöglichen eine weitere Einschränkung auf einen gewissen XML-Kontext, in welchem das Objekt stehen muss.

Anwendungsbeispiel: Man sucht nach allen chemischen Elementen, die eine Elektronegativität von 0 haben. Um diese zu finden, kann man wie folgt vorgehen:

- Man befüllt das erste Textfeld mit "`0`",
- man setzt die erste Radio-Button-Gruppe für den XML-Typ auf "`Character Data`",
- man befüllt das zweite Textfeld für den XML-Pfad mit "`electronegativity`",
- und man setzt die zweite Radio-Button-Gruppe auf "`XMLPath must have this String as Substring`".

Für das Absetzen der Suche betätigt man den Button "`Search`" unten im Formular. Danach werden dem Benutzer die Ergebnisse sofort im Browser präsentiert. Es werden alle Dokumente aufgelistet, die den Parametern aus der Anfrage genügen. Jeder Eintrag dieser Liste enthält einen Link, welcher auf das zugehörige Dokument verweist.

Für das oben beschriebene Beispiel wird das in Abbildung 5.2 dargestellte Suchergebnis zurückgeliefert:

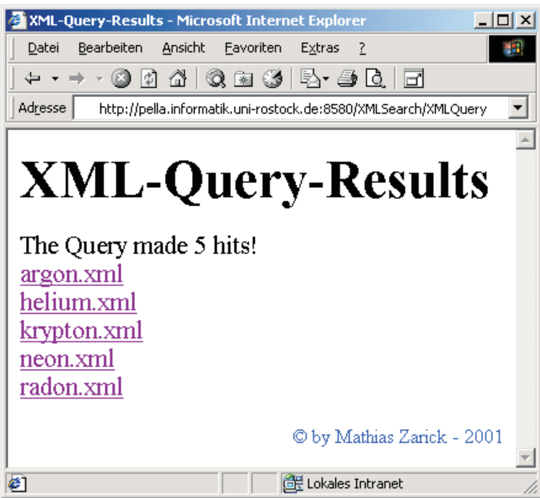
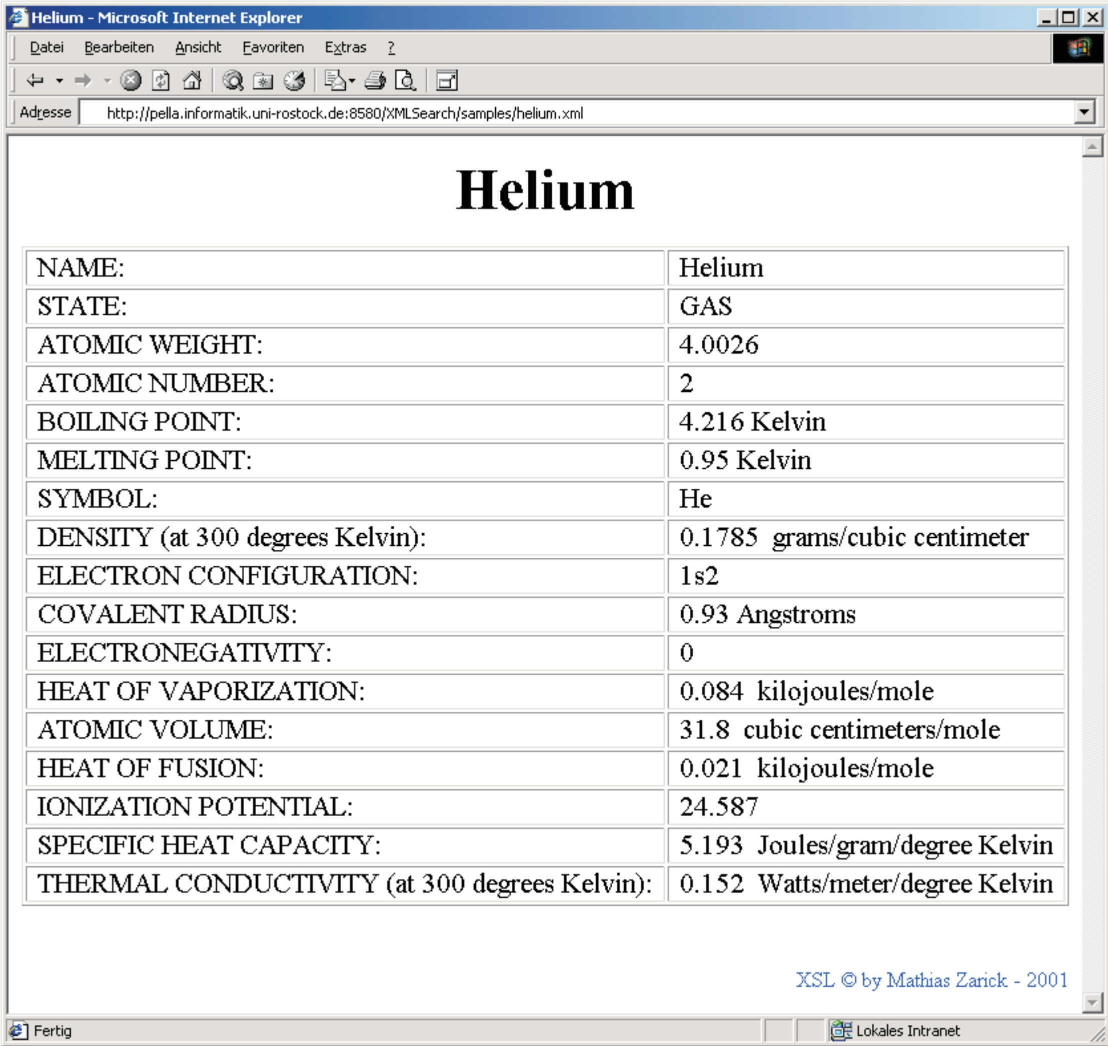


Abbildung 5.2: Ein Suchergebnis

Durch Mausklick auf die aufgelisteten Dokumente verfolgt man die Links zu den einzelnen Dokumenten. Diese werden dann im Browser präsentiert.

Falls der Browser das eingebundene XSL-Stylesheet richtig transformiert, zeigt er das Dokument `helium.xml` wie in Abbildung 5.3 dargestellt an, ansonsten wird der rohe XML-Inhalt angezeigt.



Helium

NAME:	Helium
STATE:	GAS
ATOMIC WEIGHT:	4.0026
ATOMIC NUMBER:	2
BOILING POINT:	4.216 Kelvin
MELTING POINT:	0.95 Kelvin
SYMBOL:	He
DENSITY (at 300 degrees Kelvin):	0.1785 grams/cubic centimeter
ELECTRON CONFIGURATION:	1s2
COVALENT RADIUS:	0.93 Angstroms
ELECTRONEGATIVITY:	0
HEAT OF VAPORIZATION:	0.084 kilojoules/mole
ATOMIC VOLUME:	31.8 cubic centimeters/mole
HEAT OF FUSION:	0.021 kilojoules/mole
IONIZATION POTENTIAL:	24.587
SPECIFIC HEAT CAPACITY:	5.193 Joules/gram/degree Kelvin
THERMAL CONDUCTIVITY (at 300 degrees Kelvin):	0.152 Watts/meter/degree Kelvin

XSL © by Mathias Zarick - 2001

Abbildung 5.3: Präsentation des XML-Dokumentes `helium.xml`

A Quellcode

A.1 Die Klasse XMLTypes

```
package XMLSearch;

/**
 * XML-Objekt-Typen
 *
 * @author Mathias Zarick
 */
public class XMLTypes
{
/**
 * Name eines XML-Elements
 */
public static final int ELEMENT_NAME = 0;
/**
 * Name eines XML-Attributes
 */
public static final int ATTRIBUTE_NAME = 1;
/**
 * Wert eines XML-Attributes
 */
public static final int ATTRIBUTE_VALUE = 2;
/**
 * XML-Character-Data
 */
public static final int CHARACTER_DATA = 3;
/**
 * Typ spielt keine Rolle
 */
public static final int IGNORE = 4;

} // class XMLTypes
```

A.2 Die Klasse XMLIndexInfo

```
package XMLSearch;

/**
 * Klasse beinhaltet einen Eintrag der XMLInfoList aus XMLIndexEntry.
 *
 * @author Mathias Zarick
 */
public class XMLIndexInfo
{
/**
 * Erstellt eine neue Instanz eines XMLIndexInfo-Objektes.
 */
XMLIndexInfo(String theDocumentName, String theXMLPath,
          int theXMLType)
{ documentName = theDocumentName;
  XMLPath = theXMLPath;
  XMLType = theXMLType;
}

/**
 * Name des indizierten Dokumentes
 */
public String documentName;
```

```
/**
 * gibt den XML-Pfad der indizierten Daten im Dokument an
 * dieser wird in XPATH-Notation gespeichert
 */
public String XMLPath;


/**
 * Typ des indizierten Objektes
 * siehe XMLTypes
 */
public int XMLType;

} // class XMLIndexInfo
```

A.3 Die Klasse XMLIndexEntry

```
package XMLSearch;

import java.util.ArrayList;

/**
 * Klasse beinhaltet einen Eintrag fuer den Index, welcher eine Liste
 * aus diesen Eintraegen bildet
 */
public class XMLIndexEntry
{
/**
 * Erstellt eine neue Instanz eines XMLIndexEntry-Objektes
 */
XMLIndexEntry(String theIndexee, int theNumberOfOccurances,
          ArrayList theXMLIndexInfoList)
{ indexee = theIndexee;
  numberOfOccurances = theNumberOfOccurances;
  XMLIndexInfoList = theXMLIndexInfoList;
}

/**
 * zu indizierendes Objekt
 */
public String indexee;

/**
 * Anzahl der Vorkommen in den Dokumenten
 */
public int numberOfOccurances;

/**
 * Liste von XMLIndexInfo-Instanzen, siehe Klasse XMLIndexInfo
 */
public ArrayList XMLIndexInfoList;

} // class XMLIndexEntry
```

A.4 Die Klasse XMLPathQueryMethods

```
package XMLSearch;

/**
 * XMLPfad-Anfrage-Methoden
 *
 * @author Mathias Zarick
 */
public class XMLPathQueryMethods
{
/**
 * Der XML-Pfad muss genau uebereinstimen.
 */
public static final int EXACT_MATCH = 0;
/**
 * Der XML-Pfad muss mit der angegebenen Phrase beginnen.
 */
public static final int PARTIAL_MATCH_START = 1;
/**
 * Der XML-Pfad muss die angegebene Phrase als Substring beinhalten.
 */
public static final int PARTIAL_MATCH_MIDDLE = 2;
```

```
/**
 * Der XML-Pfad muss mit der angegebenen Phrase enden.
 */
public static final int PARTIAL_MATCH_END = 3;
/**
 * Der XML-Pfad spielt bei der Suche keine Rolle.
 */
public static final int IGNORE = 4;

} // class XMLPathQueryMethods
```

A.5 Die Klasse XMLIndex

```
package XMLSearch;

import java.util.ArrayList;
import java.io.File;

/**
 * Klasse zum speichern der indizierten Daten
 *
 * @author Mathias Zarick
 */

public class XMLIndex extends ArrayList
{

/**
 * Einfuegen von Daten in den Index
 * documentName .. Name des indizierten Dokumentes
 * indexee .. Name des indizierten Objektes
 * XMLPath .. XML-Pfad des indizierten Objektes
 * XMLType .. Typ des indizierten Objektes (Elementname, Attributname,
 *         Attributwert, Character Data)
 */
public void add(String documentName, String indexee,
            String XMLPath, int XMLType)
{ XMLIndexInfo xii = new XMLIndexInfo(documentName, XMLPath, XMLType);
  int i=0;
  while (i <= this.size())
  { if (i == this.size())
    { // am Ende der Liste --> Objekt war noch nicht vorhanden --> erstellen
      ArrayList XMLIndexInfoList = new ArrayList();
      XMLIndexInfoList.add(xii);
      XMLIndexEntry newxie = new XMLIndexEntry(indexee,1,XMLIndexInfoList);
      this.add(newxie);
      break;
    }
    else
    { XMLIndexEntry xie = (XMLIndexEntry)this.get(i);
      int compare = indexee.compareToIgnoreCase(xie.indexee);
      if (compare == 0)
      { // Eintrag fuer dieses Objekt schon vorhanden --> erweitern
        xie.numberOfOccurances++;
        xie.XMLIndexInfoList.add(xii);
        break;
      }
      if (compare < 0)
      { // Eintrag fuer dieses Objekt noch nicht vorhanden --> erstellen
        ArrayList XMLIndexInfoList = new ArrayList();
        XMLIndexInfoList.add(xii);
        XMLIndexEntry newxie = new XMLIndexEntry(indexee,1,XMLIndexInfoList);
        this.add(i,newxie);
        break;
      }
    }
    i++;
  }
}

/**
 * Die benutzten ArrayList-Objekte auf die Groesse trimmen, die sie benutzen,
 * um unnoetig benutzten Speicher freizugeben.
 */
public void trim()
{ this.trimToSize();
  for (int i=0; i<this.size(); i++)
  { XMLIndexEntry xie = (XMLIndexEntry)this.get(i);
    xie.XMLIndexInfoList.trimToSize();
  }
}
```

```
/**
 * Den Index aus einer Datei einlesen.
 * NICHT IMPLEMENTIERT!
 */
public void readFromFile(File file)
{ System.out.println("readFromFile");
}

/**
 * Den Index in eine Datei schreiben.
 * NICHT IMPLEMENTIERT!
 */
public void writeToFile(File file)
{ System.out.println("writeToFile");
}

/**
 * gibt an, ob subString in mainString enthalten ist
 */
private boolean contains(String mainString, String subString)
{ boolean result = false;
  for (int i=0; i < mainString.length(); i++)
    if (mainString.startsWith(subString,i))
     { result = true;
       break;
     }
  return result;
}

/**
 * Divide-and-Conquer-Algorithmus zum Finden der zu suchenden Phrase im Index
 */
public ArrayList query(String phrase, String XMLPath,
                  int queryMethod, int XMLType)
{
ArrayList result = new ArrayList();
if ((this.size() > 0) && (phrase!=null))
{ int searchRangeBegin = 0;
  int searchRangeEnd = this.size()-1;
  boolean found = false;
  boolean end = false;
  while (!end && !found)
  { end = (searchRangeBegin >= searchRangeEnd);
    XMLIndexEntry xie =
      (XMLIndexEntry)this.get((searchRangeBegin + searchRangeEnd) / 2);
    int compare = phrase.compareToIgnoreCase(xie.indexee);
    if (compare == 0)
    { found = true;
      for (int i=0; i < xie.numberOfOccurances; i++)
      { XMLIndexInfo xii = (XMLIndexInfo)xie.XMLIndexInfoList.get(i);
        if (XMLPath != null)
         { boolean condition = false;
           switch (queryMethod) {
           case XMLPathQueryMethods.EXACT_MATCH:
             condition = xii.XMLPath.equalsIgnoreCase(XMLPath);
           break;
           case XMLPathQueryMethods.PARTIAL_MATCH_START:
             condition =
               xii.XMLPath.toLowerCase().startsWith(XMLPath.toLowerCase());
           break;
           case XMLPathQueryMethods.PARTIAL_MATCH_MIDDLE:
             condition =
               contains(xii.XMLPath.toLowerCase(), XMLPath.toLowerCase());
           break;
           case XMLPathQueryMethods.PARTIAL_MATCH_END:
             condition =
               xii.XMLPath.toLowerCase().endsWith(XMLPath.toLowerCase());
           break;
           case XMLPathQueryMethods.IGNORE:
             condition = true;
           break;
           }
           if (condition
               && ((XMLType == XMLTypes.IGNORE) || (XMLType == xii.XMLType))
               && !result.contains(xii.documentName))
             result.add(xii.documentName);
         }
         else if (((XMLType == XMLTypes.IGNORE) || (XMLType == xii.XMLType))
                && !result.contains(xii.documentName))
             result.add(xii.documentName);
      }
    }
```

```
    if (compare < 0)
    { searchRangeEnd = ((searchRangeBegin + searchRangeEnd) / 2) - 1;
    }
    if (compare > 0)
    { searchRangeBegin = ((searchRangeBegin + searchRangeEnd) /2 ) + 1;
    }
  }
}
return result;
}

} // class XMLIndex
```

A.6 Die Klasse XMLQuery

```
package XMLSearch;

/* Studienarbeit
   Konzeption einer Suchmaschine fuer XML-Dokumente
   Mathias Zarick
   April 2001
*/

import java.io.IOException;
import java.io.PrintWriter;
import java.io.File;
import java.io.FileOutputStream;

import java.util.Arrays;
import java.util.ArrayList;
import java.util.ResourceBundle;
import java.util.MissingResourceException;
import java.util.Date;
import java.util.Enumeration;

import java.text.SimpleDateFormat;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;


/**
 * Servlet fuer die XML-Suchmaschine
 *
 * @author Mathias Zarick
 */

public class XMLQuery extends HttpServlet {


   private static ResourceBundle rb;

   /* siehe XMLSearch.properties */
   private static String DEFAULT_PARSER_NAME;
   private static String SEARCH_DIR;
   private static String PRESENTATION_DIR;
   private static String QUERY_LOGFILE;
   private static boolean setValidation;
   private static boolean setNameSpaces;
   private static boolean setSchemaSupport;

   /* Die Index-Struktur */
   private static XMLIndex index;

   /*Die XML-Parser-Instanz */
   private static XMLParse xmlparse;

   private static PrintWriter logging;
   private static SimpleDateFormat sdf;
```

```
/**
 * Servlet-GET-Methode
 * request .. ankommende Daten vom Browser
 * response .. abzusendende Daten an den Browser
 */
  public void doGet(HttpServletRequest request,
                 HttpServletResponse response)
     throws IOException, ServletException
  {
     response.setContentType("text/html");

     PrintWriter out = response.getWriter();
     out.println("<html>");
     out.println("<head>");

     String title = "XML-Query-Results";
     out.println("<title>" + title + "</title>");
     out.println("</head>");
     out.println("<body bgcolor=\"white\">");

     out.println("<h1>" + title + "</h1>");

     String phrase = request.getParameter("phrase");
     String XMLTypeString = request.getParameter("XMLType");
     int XMLType = XMLTypes.IGNORE;
     if (XMLTypeString != null)
           if (XMLTypeString.equals("ELEMENT_NAME"))
                 XMLType = XMLTypes.ELEMENT_NAME;
       else if (XMLTypeString.equals("ATTRIBUTE_NAME"))
                 XMLType = XMLTypes.ATTRIBUTE_NAME;
       else if (XMLTypeString.equals("ATTRIBUTE_VALUE"))
                 XMLType = XMLTypes.ATTRIBUTE_VALUE;
       else if (XMLTypeString.equals("CHARACTER_DATA"))
                 XMLType = XMLTypes.CHARACTER_DATA;
     String XMLPath = request.getParameter("XMLPath");
     String XMLPathQueryMethodString = request.getParameter("XPQM");
     int XMLPathQueryMethod = XMLPathQueryMethods.IGNORE;
     if (XMLPathQueryMethodString != null)
           if (XMLPathQueryMethodString.equals("EXACT_MATCH"))
                 XMLPathQueryMethod = XMLPathQueryMethods.EXACT_MATCH;
       else if (XMLPathQueryMethodString.equals("PARTIAL_MATCH_START"))
                 XMLPathQueryMethod = XMLPathQueryMethods.PARTIAL_MATCH_START;
       else if (XMLPathQueryMethodString.equals("PARTIAL_MATCH_MIDDLE"))
                 XMLPathQueryMethod = XMLPathQueryMethods.PARTIAL_MATCH_MIDDLE;
       else if (XMLPathQueryMethodString.equals("PARTIAL_MATCH_END"))
                 XMLPathQueryMethod = XMLPathQueryMethods.PARTIAL_MATCH_END;

     ArrayList queryResult = index.query(phrase, XMLPath, XMLPathQueryMethod, XMLType);

     out.println("The Query made " + queryResult.size() + " hits!");
     out.println("<br>");
     for (int i=0; i < queryResult.size(); i++)
     { String oneResult = (String)queryResult.get(i);
      out.println("<a href=\"" + PRESENTATION_DIR + "/" + oneResult + "\">" + oneResult + "</a><br>");
     }
     out.println("<p align=\"right\"><font size=\"-3\" color=\"blue\">&copy; by Mathias Zarick - 2001</p>");
     out.println("</body>");
     out.println("</html>");

     logging.println("Date: " + sdf.format(new Date(System.currentTimeMillis())));
     Enumeration e = request.getHeaderNames();
     while (e.hasMoreElements()) {
        String name = (String)e.nextElement();
        String value = request.getHeader(name);
        logging.println(name + ": " + value);
     }
     logging.println("Method: " + request.getMethod());
     logging.println("Request URI: " + request.getRequestURI());
     logging.print("Protocol: " + request.getProtocol());
     logging.println("PathInfo: " + request.getPathInfo());
     logging.println("Remote Address: " + request.getRemoteAddr());

     logging.println("phrase: " + phrase);
     logging.println("XMLPath: " + XMLPath);
     logging.print("XMLType: ");logging.println(XMLType);
     logging.print("XMLPathQueryMethod: ");logging.println(XMLPathQueryMethod);
     logging.println("--------------------------------");
     logging.flush();
  }
```

```
/**
 * Servlet-POST-Methode
 * request .. ankommende Daten vom Browser
 * response .. abzusendende Daten an den Browser
 */
   public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
       throws IOException, ServletException
   {
       doGet(request, response);
   }

/**
 * Initialisiert das Servlet, dabei wird das Properties-File XMLSearch.properties
 * ausgelesen und dementsprechend ein Index aufgebaut
 */
   public void init() throws ServletException
   { boolean error = false;
     try {
       rb = ResourceBundle.getBundle("XMLSearch.XMLSearch");

       DEFAULT_PARSER_NAME = rb.getString("DEFAULT_PARSER_NAME");
       SEARCH_DIR = rb.getString("SEARCH_DIR");
       PRESENTATION_DIR = rb.getString("PRESENTATION_DIR");
       QUERY_LOGFILE = rb.getString("QUERY_LOGFILE");

       String setValidationString = rb.getString("setValidation");
       setValidation = (setValidationString.equals("true"));

       String setNameSpacesString = rb.getString("setNameSpaces");
       setNameSpaces = (setNameSpacesString.equals("true"));

       String setSchemaSupportString = rb.getString("setSchemaSupport");
       setSchemaSupport = (setSchemaSupportString.equals("true"));
      }
      catch (MissingResourceException mre)
      { System.err.println("Cannot load a correct Properties-File \"XMLSearch.properties\""
                  + " in the Classpath of Package XMLSearch!");
       error = true;
      }
       sdf = new SimpleDateFormat();
       try { logging = new PrintWriter(new FileOutputStream(QUERY_LOGFILE));
            logging.println("Servlet started - " + sdf.format(new Date(System.currentTimeMillis())));
            logging.println("--------------------------------");
            logging.flush();
       }
       catch (java.io.IOException e)
       { System.err.println("Error writing to Logfile: '" + QUERY_LOGFILE + "'!");
       }
       index = new XMLIndex();
       if (!error)
       { System.out.println("Generating Index!");
         File dir = new File(SEARCH_DIR);
         if (dir.isDirectory())
         { String[] fileList = dir.list();
           Arrays.sort(fileList);
           for (int i = 0; i < fileList.length; i++) {
             String fileString = fileList[i];
             File file = new File(fileString);
             if (!file.isDirectory() && fileString.toLowerCase().endsWith(".xml"))
               { xmlparse = new XMLParse(fileString);
                 xmlparse.parse(DEFAULT_PARSER_NAME, SEARCH_DIR + "/" + fileString, setValidation);
               }
           }
         }
         else System.err.println(SEARCH_DIR + " not found!");
       }
       index.trim();
   }


private class XMLParse
extends DefaultHandler {

XMLParse(String theDocument)
{ document = theDocument;
}

   /** current XML-Document. */
   private String document;

   /** current XML-Path. */
   private String XMLPath;
```

```java
public void parse(String parserName, String uri, boolean validate) {

   try {
      System.out.println("Processing "+uri+"!");

      XMLReader parser = (XMLReader)Class.forName(parserName).newInstance();
      parser.setContentHandler(xmlparse);
      parser.setErrorHandler(xmlparse);


      parser.setFeature( "http://xml.org/sax/features/validation",
                          validate);

      parser.setFeature( "http://xml.org/sax/features/namespaces",
                          setNameSpaces );

      parser.setFeature( "http://apache.org/xml/features/validation/schema",
                          setSchemaSupport );

      parser.parse(uri);
   } catch (org.xml.sax.SAXParseException spe) {
      spe.printStackTrace(System.err);
   } catch (org.xml.sax.SAXException se) {
      if (se.getException() != null)
         se.getException().printStackTrace(System.err);
      else
         se.printStackTrace(System.err);
   } catch (Exception e) {
      e.printStackTrace(System.err);
   }

}

//
// DocumentHandler methods
//

/** Start document. */
public void startDocument() {
XMLPath = "/";
}

/** Start element. */
public void startElement(String uri, String local, String raw, Attributes attrs) {
   index.add(document,raw,XMLPath,XMLTypes.ELEMENT_NAME);
   XMLPath = XMLPath + raw + "/";
   if (attrs != null)
    for (int i=0; i < attrs.getLength(); i++) {
      index.add(document,attrs.getQName(i),XMLPath,XMLTypes.ATTRIBUTE_NAME);
      index.add(document,attrs.getValue(i),XMLPath+"@"+attrs.getQName(i)+"/",XMLTypes.ATTRIBUTE_VALUE);
     }
}

/** End element. */
public void endElement(String uri, String local, String raw) {
   XMLPath = XMLPath.substring(0,XMLPath.length()-raw.length()-1);
}

/** Characters. */
public void characters(char ch[], int start, int length) {
   String chars = new String(ch, start, length).trim();
   if (!chars.equals("")) index.add(document, chars, XMLPath, XMLTypes.CHARACTER_DATA);
}

//
// ErrorHandler methods
//

/** Warning. */
public void warning(SAXParseException ex) {
   System.err.println("[Warning] "+
                  getLocationString(ex)+": "+
                  ex.getMessage());
}

/** Error. */
public void error(SAXParseException ex) {
   System.err.println("[Error] "+
                  getLocationString(ex)+": "+
                  ex.getMessage());
}
```

```
   /** Fatal error. */
   public void fatalError(SAXParseException ex) throws SAXException {
      System.err.println("[Fatal Error] "+
                   getLocationString(ex)+": "+
                   ex.getMessage());
   }

   /** Returns a string of the location. */
   private String getLocationString(SAXParseException ex) {
      StringBuffer str = new StringBuffer();
      String systemId = ex.getSystemId();
      if (systemId != null) {
         int index = systemId.lastIndexOf('/');
         if (index != -1)
            systemId = systemId.substring(index + 1);
         str.append(systemId);
      }
      str.append(':');
      str.append(ex.getLineNumber());
      str.append(':');
      str.append(ex.getColumnNumber());
      return str.toString();
   }

} // class XMLParse

} // class XMLQuery
```

A.7 Das Anfrageformular index.html

```
<HTML>
<!-- Studienarbeit
   Konzeption einer Suchmaschine fuer XML-Dokumente
   Mathias Zarick
   April 2001
-->
  <HEAD>
      <TITLE>XML Query Form</TITLE>
  </HEAD>
    <BODY BGCOLOR="#FFFFFF" TEXT="#000000">
      <H1>XML Querying</H1>
      <BR>
      <FORM NAME="XMLQUERY" ACTION="/XMLSearch/XMLQuery" METHOD="POST">
         <TABLE WIDTH="100%" CELLSPACING="0">
            <TR VALIGN="BOTTOM">
               <TD WIDTH="55%">
                Search for:
                <BR>
                <INPUT NAME="phrase" MAXLENGTH="255" SIZE="30">
               </TD>
               <TD WIDTH="45%">
               <INPUT TYPE="RADIO" NAME="XMLType" VALUE="ELEMENT_NAME"> Elementname
               <BR>
               <INPUT TYPE="RADIO" NAME="XMLType" VALUE="ATTRIBUTE_NAME"> Attributename
               <BR>
               <INPUT TYPE="RADIO" NAME="XMLType" VALUE="ATTRIBUTE_VALUE"> Attributevalue
               <BR>
               <INPUT TYPE="RADIO" NAME="XMLType" VALUE="CHARACTER_DATA"> Character Data
               <BR>
               <INPUT TYPE="RADIO" NAME="XMLType" CHECKED VALUE="IGNORE"> Anything of that
               </TD>
            </TR>
         </TABLE>
      <BR>
      <BR>
      <BR>
         <TABLE WIDTH="100%" CELLSPACING="0">
            <TR VALIGN="BOTTOM">
               <TD WIDTH="55%">
                Specify an XMLPath, which the
                <BR>
                search phrase shall be underlain!
                <BR>
                <INPUT NAME="XMLPath" MAXLENGTH="255" SIZE="30">
               </TD>
               <TD WIDTH="45%">
               <INPUT TYPE="RADIO" NAME="XPQM" VALUE="EXACT_MATCH"> XMLPath must be exactly this String
               <BR>
               <INPUT TYPE="RADIO" NAME="XPQM" VALUE="PARTIAL_MATCH_START"> XMLPath must start with this String
               <BR>
               <INPUT TYPE="RADIO" NAME="XPQM" VALUE="PARTIAL_MATCH_MIDDLE"> XMLPath must have this String as Substring
               <BR>
```

```
            <INPUT TYPE="RADIO" NAME="XPQM" VALUE="PARTIAL_MATCH_END"> XMLPath must end with this String
            <BR>
            <INPUT TYPE="RADIO" NAME="XPQM" CHECKED VALUE="IGNORE"> XMLPath does not matter
            </TD>
          </TR>
        </TABLE>
        <BR>
        <BR>
        <BR>
      <INPUT TYPE="SUBMIT" VALUE="   Search   ">      
      <INPUT TYPE="RESET" VALUE="Clear Form">
      </FORM>
      <BR>
      <P ALIGN="RIGHT"><FONT SIZE="-3" COLOR="BLUE">&copy; by Mathias Zarick - 2001</FONT></P>
    </BODY>
</HTML>
```

A.8 Die Konfigurations-Datei XMLSearch.properties

```
# $Id: XMLSearch.properties $

# Der zu nutzende XML-SAX-Parser
DEFAULT_PARSER_NAME=org.apache.xerces.parsers.SAXParser

# Pfad mit den zu indizierenden XML-Dateien
# Angabe relativ zu bin-Verzeichnis von Tomcat
# oder absolut
SEARCH_DIR=../webapps/XMLSearch/samples
# SEARCH_DIR=D:/Programme/jakarta-tomcat-3.2.1/webapps/XMLSearch/samples

# relativer Pfad zu den XML-Dateien (aus der Sicht des Pfades des Servlet-Formulars index.html)
# oder absolut auf dem Tomcat-Webserver
PRESENTATION_DIR=samples
# PRESENTATION_DIR=/XMLSearch/samples

# Datei fuer das Loggen der statistischen Informationen
# was wurde wann abgefragt
QUERY_LOGFILE=../webapps/XMLSearch/query.log

# DTD und Schema Validierung an oder aus
setValidation=true

# NamesSpaces-Support an oder aus
setNameSpaces=true

# Schema-Support an oder aus
setSchemaSupport=false
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler.
Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, 6 October 2000,
http://www.w3.org/TR/REC-xml

[2] Steve DeRose, Eve Maler, David Orchard.
XML Linking Language (XLink) Version 1.0, W3C Proposed Recommendation, 20 December 2000,
http://www.w3.org/TR/xlink

[3] XYZFind Corporation.
XYZFind Server User's Guide, Version 1.01, March 2001,
http://www.xyzfind.com/download/docs/userguide.pdf

[4] James Clark, Steve DeRose.
XML Path Language (XPath) Version 1.0, W3C Recommendation, 16 November 1999,
http://www.w3.org/TR/xpath

[5] David C. Fallside.
XML Schema Part 0: Primer, W3C Proposed Recommendation, 16 March 2001,
http://www.w3.org/TR/xmlschema-0

[6] Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn.
XML Schema Part 1: Structures, W3C Proposed Recommendation, 16 March 2001,
http://www.w3.org/TR/xmlschema-1

[7] Paul V. Biron, Ashok Malhotra.
XML Schema Part 2: Datatypes, W3C Proposed Recommendation, 16 March 2001,
http://www.w3.org/TR/xmlschema-2

[8] The Infozone Group.
SchemoX Project,
http://www.infozone-group.org/schemoxDocs/html/proposal.html

[9] Don Chamberlin, Daniela Florescu, Jonathan Robie, Jérôme Siméon, Mugur Stefanescu.
XQuery: A Query Language for XML, W3C Working Draft, 15 February 2001,
http://www.w3.org/TR/xquery

[10] Jonathan Robie, Joe Lapp, David Schach.
XML Query Language(XQL), September 1998
http://www.w3.org/TandS/QL/QL98/pp/xql.html

[11] Apache Software Foundation.
Tomcat
http://jakarta.apache.org/tomcat

[12] ibiblio.org.
http://www.ibiblio.org/xml/examples/periodic_table/allelements.xml

[13] James Clark.
XSL Transformations (XSLT) Version 1.0, W3C Recommendation, 16 November 1999,
http://www.w3.org/TR/xslt.html

[14] Apache Software Foundation.
Xerces Java Parser 1.3.1 Release
http://xml.apache.org/xerces-j

[15] Ian H. Witten, Alistair Moffat, Timothy C. Bell.
Managing Gigabytes: Compressing and Indexing Documents and Images, Second Edition, May 1999

[16] K. J. McDonell.
An inverted index implementation, August 1975,
http://www3.oup.co.uk/jnls/list/computer_journal/hdb/Volume_20/Issue_02/200116.sgm.abs.html