DB-orientierte Aufbereitung und Suchunterstützung für Lehr-/Lerndokumente

Marcus Flehmig, Theo Härder
Fachbereich Informatik, Universität Kaiserslautern,
Postfach 3049, 67653 Kaiserslautern,
{flehmig|haerder}@informatik.uni-kl.de
http://wwwdbis.informatik.uni-kl.de

1. Einführung

Um mit der künftigen Entwicklung auf dem Gebiet des netzbasierten Lehren und Lernens Schritt halten zu können, bedarf es vermehrter Anstrengungen zur Nutzung und Integration der neuen Informations- und Kommunikationstechnologien. Durch das Internet wird ohne zusätzliche Kosten eine Zeit-, Plattform- und Ortsunabhängigkeit des Lernens erreicht, wenn die Lehr-/Lerninhalte in so genannten Web-Informationssystemen auf der Basis von rechnergestützten multimedialen Netzwerken verfügbar gemacht werden. Dabei lassen sich zugleich neue Formen des Lehrens und Lernens verwirklichen, die bisher im Fernstudium nicht zu realisieren waren.

2. Web-Technologien: Realisierte Anwendungen

Prüfungsinformationssysteme: PAS und PAVIAN

Die Systeme PAS (Prüfungs-Anmelde-System) und PAVIAN (Prüfungs-Administration VIA Netz) ermöglichen eine Web-basierte Anmeldung aller Studenten zu Prüfungen, die Erstellung, Verwaltung und Auswertung von Prüfungslisten für den Bereich der Dekanatsverwaltung sowie die Aufbereitung einer Vielfalt von Listen für verschiedene Abläufe in der Prüfungsverwaltung. Sie befinden sich erfolgreich innerhalb des Fachbereichs Informatik im Einsatz und gestatten eine immer weitergehende Automatisierung vieler Routineaufgaben. Anfänglich nur in einem Studiengang eingesetzt, unterstützen die Systeme nunmehr die Verwaltung von drei unterschiedlichen Informatik-Studiengängen.

Metadatenzugang für akademisches Lehr- und Lernmaterial: Meta-Akad

Über das Internet werden von Hochschulen, Bildungseinrichtungen und Verlagen im In- und Ausland Lehr- und Lernmaterialien zur Verfügung gestellt. Diese Angebote sind jedoch meistens nur lokal bekannt. Das Projekt META-AKAD[1] hat das Ziel, Lernenden und Lehrenden einen einheitli-

chen Zugang zu einem umfassenden Angebot von im Internet verfügbaren Lehr- und Lernmaterial zu verschaffen. Der Zugang wird als Web-basierte virtuelle Bibliothek realisiert werden. Dabei werden verschiedene Aspekte der Entwicklung eines solchen Dienstes gleichermaßen intensiv betrieben und integriert:

- Aufbau einer umfangreichen und repräsentativen Sammlung von Lerndokumenten,
- Erschließung der Objekte mit einem elaborierten Metadatensatz und bibliothekarischen Methoden,
- Inhaltliche und didaktische Qualitätsbeurteilung und Auswahl des Materials,
- Moderne und zukunftssichere Systemarchitektur,
- Benutzerorientierte Entwicklung.

Im Projekt Meta-Akad kommen viele der hier vorgestellten Konzepte, Verfahren und Ansätze zum tatsächlichen Einsatz.

3. Individualisierung / Personalisierung

Portale haben sich zu der Basistechnologie für E-Learning-Plattformen entwickelt [Fle01b]. Das Projekt SHARX zur Realisierung des bereits vorgestellten Unified-View-Konzeptes [Fle01a] stellt eine Kombination von

- deklarativem Web-Site Management,
- Integration heterogener Datenquellen,
- Portal-Architekturen und
- XML-Technologien

zwecks Untersuchung und prototypischer Realisierung umfassender Personalisierungskonzepte dar. Die Verwirklichung des Konzeptes eines Unified View zwecks Bilden einer übergreifenden Sicht durch generische Integrationansätze erfordert eine genaue Betrachtung der verschiedenen Stufen während der Integration der zugrundeliegenden Daten.

1.) http://rzblx1.uni-regensburg.de/metaakad/homepage/

Die Anforderung an ein solches umfassendes Personalisierungskonzept lassen sich wie folgt umreissen:

- Die zugrundeliegenden Daten werden benutzerseitig in ihrer Gesamtheit (Unified View) betrachtet.
- Es ist möglich, auf der Basis des Unified View zu selektieren,
- zu restrukturieren und
- die Inhalte individuell zu präsentieren.
- Unterstützung von „*Multi-Channel Delivery*“ (MCD).

Ein umfassendes Konzept, wie oben angedeutet, erfordert eine integrierte Sicht auf die Gesamtheit aller Daten. Um ferner auch Restrukturierung zu ermöglichen, ist die Trennung von zugrundeliegenden Daten, Strukturen und Repräsentationen unabdingbar. Aus der Möglichkeit der Unterstützung verschiedener Endgeräte ergibt sich des Weiteren die Anforderung zwischen eigentlicher Personalisierung und dem so genannten „Customization“ zu unterscheiden [CFP99].

3.1 Realisierungsaspekte eines umfassenden Personalisierungskonzeptes

Existierende Web-basierende Informationssysteme orientieren sich im Allgemeinen an Dokumentenoperationen, während XML-Anwendungen Datenbankoperationen benötigen [Suc98]. Die Allgemeingültigkeit von XML verwischt die Unterschiede zwischen Daten und Dokumenten, so dass sich zwei Betrachtungsweisen entwickelt haben. Der einen liegt eine dokumentenorientierte, der anderen eine datenbankorientierte Sichtweise zugrunde [ABS00]. Mit dem ersten Fall ist eine textuelle Sicht auf Dokumente als Ganzes, beispielsweise im Anwendungsgebiet des elektronischen Publizierens (POP: Presentation-Oriented Publishing) von Manuskripten, Büchern oder strukturierten Texten, mit dem zweiten Fall hingegen, beispielsweise im Bereich datenintensiver Web-Anwendungen oder Message-Oriented Middleware (MOM), eine datenorientierte Sicht verbunden. In beiden Fällen aber werden Daten, im allgemeinsten Sinne, durch eine Menge von XML-Dokumenten repräsentiert.

3.2 Das Web-Site-Management-System SHARX

Das Kernsystem von SHARX realisiert die Verarbeitungsschicht, d. h., neben der externen, konzeptionellen, internen Schicht und deren Support-Schichten (XML-Personalizer, XML-Integrator, Partition-Miner) sind die Autorisierung und Konfiguration ebenso hier angesiedelt. Jedes Datenverwaltungssystem bildet zusammen mit ihrem jeweiligen XML-Wrapper einen so genannten „XML Data Provider“. XML Data Provider sind autonome Datenverwaltungssysteme, die ihre Daten durch eine in SHARX spezifizierte XML-basierte Schnittstelle anbieten. Das Ziel der Anbindung der jeweiligen Datenquelle besteht darin, die Eigenschaften und Möglichkeiten der zugehörigen

Datenverwaltungssysteme zu berücksichtigen und, wenn möglich, auch zu nutzen. Im Folgenden wird das Kernsystem näher erläutert und detaillierter auf die Realisierung der jeweiligen Komponenten eingegangen.

3.2.1 Kernsystem

Aus Abb. 3.1 kann man erkennen, dass das Kernsystem aus den Komponenten XML-Personalizer, Mediator, XML-Integrator, Partition-Miner, Registry, Repository, Authority und Configuration besteht. Der XML-Personalizer ist der Ebene der externen Schicht zuzuordnen. Die Mediator-Komponente realisiert den Unified View, wobei der XML-Integrator und der Partition-Miner seiner Erstellung dienen. Die Ebene der internen Schicht wird realisiert durch die Registry- und Repository-Komponente. Auf der Ebene der externen Schicht soll es möglich sein, deklarativ Sichten basierend auf dem Unified View formulieren zu können.

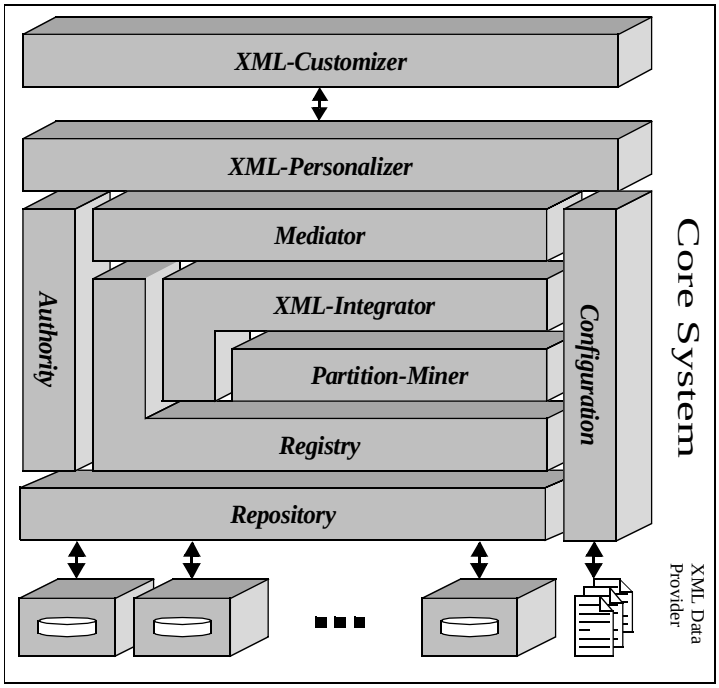


Abb. 3.1: SHARX Architektur

Die den Unified View realisierende Mediator-Komponente muss daher in der Lage sein, Anfragen zu verarbeiten. Teilanfragen können entsprechend der Eigenschaften und Möglichkeiten der XML Data Provider auch an diese delegiert werden. Besitzt ein XML Data Provider nicht die Fähigkeit, Anfragen zu beantworten, so ist diese Verarbeitung in der Mediator-Komponente durchzuführen. Es ist hierbei durchaus vorstellbar, dass die entstandenen Zwischenergebnisse über die Dauer einer Anfrage hinausgehend vorgehalten werden (Caching) oder Anfrageergebnisse aus vorgenerierten XML-Fragmenten beantwortet werden. Die Entscheidung zwischen dynamischer Anfrageverarbeitung und die Kombination mit statisch vorgenerierten Teile obliegt der Komponente XML Integrator. Sie ist verantwortlich für die Ausführung und Auswahl der Integrationsstrategie unter Berücksichtigung der Eigenschaften und Möglichkeiten der jeweiligen XML Data Provider. Der Partition-Miner ist, wie bereits erwähnt, für die Analyse der Dokumentenbasis zuständig. Dies geschieht auf der Grundlage der Menge von XML-Dokumenten, weshalb es wichtig ist, eine geeignete Unterstützung hierfür vorzusehen. Diese Unterstützung durch die flexible Kopplung mittels XML Data Provider wurde bereits vorgestellt [Sie01]. Die darauf aufbauenden Schichten, die bereits untersucht wurden, werden im Folgenden vorgestellt.

3.2.2 Repository

Die Repository-Komponente ist zwischen den XML Data Provider und dem übrigen XML-Dateien verarbeitenden System angeordnet. Sie kommuniziert mit den XML Data Provider, empfängt An- und Abmeldungen von ihnen, und unterhält ein Verzeichnis sämtlicher Dokumente, die zur Verfügung gestellt werden. Das Repository stellt nach oben die von den XML Data Provider kommenden Dokumente zur Verfügung und koordiniert den Zugriff auf diese. Das Repository hat einige Grundaufgaben zu erfüllen, die im Folgenden genannt werden sollen:

- Dokument-Verwaltung: Der Dokument-Inhalt kann über die Repository-Schnittstelle angefragt werden. Es können neue Dokumente angelegt werden. Vorhandene Dokumente können geändert oder gelöscht werden, sofern der entsprechende Wrapper dies zulässt. Die Zugriffsrechte für ein Dokument können über Proxy-Objekte erfragt werden.
- RMI-Server: Das Repository unterhält einen RMI-Server, an dem sich XML Data Provider an- und abmelden können.

Die Kommunikation zwischen Repository und XML Data Provider wurde zunächst mittels RMI[2] realisiert. Das Repository stellt also einen RMI-Server mit der Metadaten-Schnittstelle zur Verfügung. RMI ist zunächst das einzige Protokoll, über das die Repository-Komponente für den Wrapper erreichbar ist. Zum restlichen System stellt das Repository eine vielseitige, universelle Schnittstelle zur Verfügung. Sie erlaubt dem System:

- Die Initialisierung und Konfiguration: Festlegung von Laufzeitparametern für das System (Wahl der internen Datenbank, Bekanntgabe der benötigten Referenzen etc.)
- Die Ressourcen-Verwaltung: Hinzufügen, Löschen und Anzeigen von Ressourcen.
- Den Ressourcen-Zugriff: Lesen, Schreiben und Anfragen von Ressourcen.

Weitere Details der Implementierung finden sich in [Höh02].

3.2.3 Registry

Wie bereits erwähnt findet sich die Registry zwischen Partition-Miner und Repository wieder und kann über die Configuration angesprochen werden. Sie stellt eine Registrierungsdatenbank für alle zu verwaltenden XML-Artefakte dar. Neben dieser Verwaltungsaufgabe muss die Registry aber auch Suchfunktionen für den Partition-Miner bereitstellen, der beispielsweise die vorkommenden Dokumentypen oder die zugehörigen Dokumente erfragen möchte.

Grundsätzlich muss also die Registry Proxy-Objekte auf XML-Dokumente und auf Strukturbeschreibungen verwalten. Die Strukturbeschreibungen werden dabei unterschieden in externe und interne Strukturbeschreibungen, wobei die externen Strukturbeschreibungen DTDs oder XML-

2.) http://java.sun.com/products/jdk/rmi/

Schemata sein können. Zu allen XML-Dokumenten werden interne Strukturbeschreibungen von SHARX durch den Schema-Extractor generiert, um im System mit einer einheitlichlichen Strukturrepräsentation arbeiten zu können.

Die weiteren Aufgaben bzw. Anforderungen sind:

- Erstellung und Verwalten von Registry-Einträgen,
- Bereitstellen von geeigneten Speicherungsstrukturen (Proxies) und
- Indexstrukturen,
- Parametrisierbarkeit der Registry und
- sinnvolle Reaktion auf das Propagieren von Änderungen durch die Configuration.

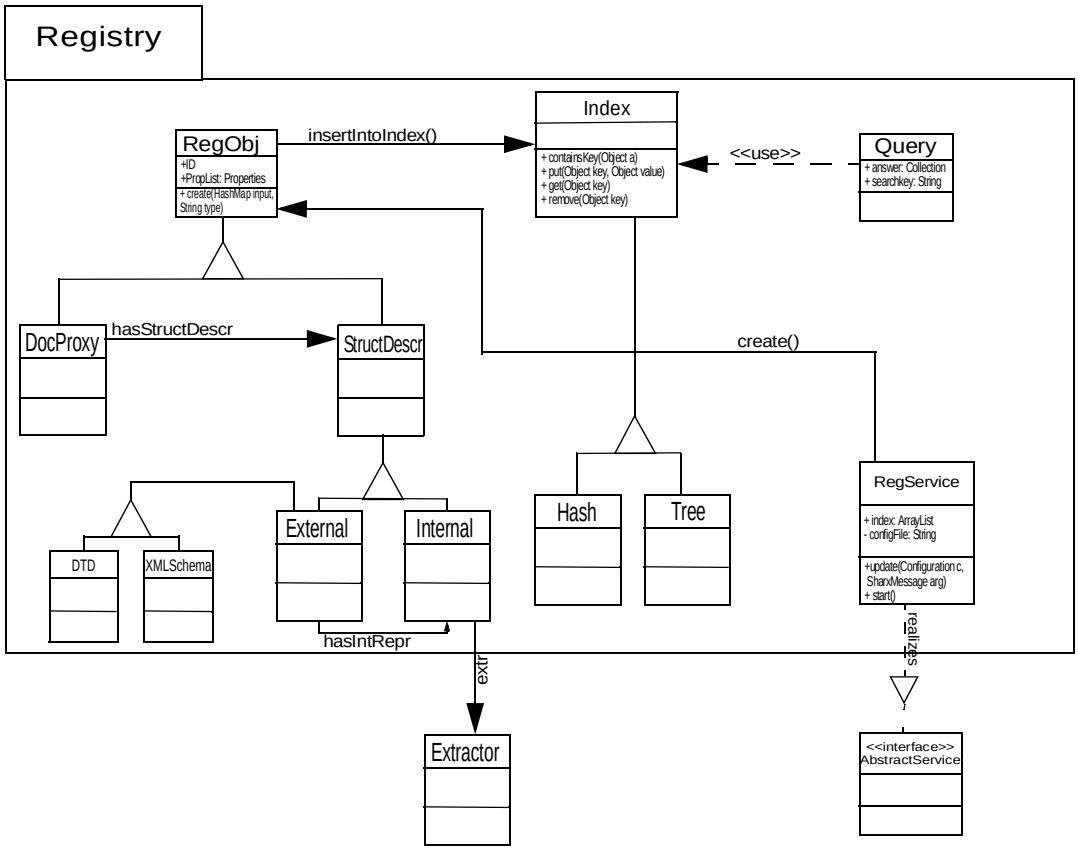


Abb. 3.2: Überblick über die XML-Artefakte und ihrer Beziehungen

Diese Anforderungen resultieren in dem in Abb. 3.2 gezeigten Entwurf, der zunächst Hauptspeicher-basiert realisiert wurde [Sch02], aber durch eine Datenbank-gestützte Lösung ersetzt werden wird.

3.2.4 Partition-Miner

Der Partition-Miner analysiert die von der Registry-Komponente verwalteten Dokumente und teilt diese in Partitionen ähnlicher Dokumente ein. Bevor die Partitionen erstellt werden, müssen eventuell die Schema-Informationen der Dokumente extrahiert werden. Die Schema-Informationen werden durch den Schema-Extractor [Büh02] extrahiert und in der Registry abgelegt. Sie ergänzen die eventuell schon vorhandenen Metadaten. Die Metadaten der Registry-Komponente werden vom XML-Integrator und vom Partition-Miner für die Integration benötigt. Das weitere Vorgehen des Partition-Miner wird in [Boh02] untersucht.

Schema-Extractor

Die Voraussetzung des Partition Mining besteht aus der Verfügbarkeit von Schemainformationen. Es soll aber auch die Klasse von Dokumenten berücksichtigt werden, die kein explizites Schema besitzen. In diesem Fall werden Schemainformationen aus den Dokumenten selbst gewonnen und mit den Informationen, die auf anderem Wege (explizite Strukturbeschreibungen: DTDs, XML-Schemata) gefunden wurden, integriert. Grob formuliert lassen sich die Anforderungen und Abgrenzungen wie folgt charakterisieren:

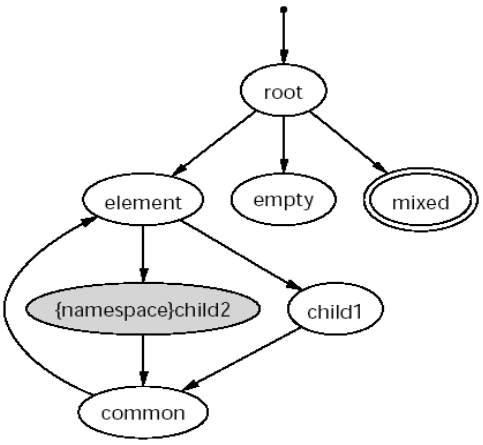
- Berücksichtigung von Namespaces,
- Sinnvolle Behandlung von Mixed Content bei Daten-orientierter Sicht auf XML-Dokumente,
- Extraktion der Schemainformationen aus XML-Dokumenten selbst,
- Ausnutzung eventuell vorhandener DTDs,
- optimierte Speicherung der Schemainformationen (Schemaintegration).

Es ist vorgesehen, dass die Schemaextraktion als zusätzlicher Dienst eingebunden wird, der die Grundlage seiner Arbeit, die zu analysierenden XML-Dokumente, über das Repository bezieht. Die Ergebnisse, also die extrahierten Schemainformationen, werden dann im Rahmen der Registry gehalten. Dienen sollen diese Ergebnisse vor allem den Komponenten Partition-Miner und Integrator, denn sie sind auf Strukturinformationen der zu integrierenden Dokumente angewiesen.

Durch die Schemaextraktion werden sie vor allem in die Lage versetzt, auch solche Dokumente zu bearbeiten, zu denen vorab keine Strukturinformationen vorhanden sind. Außerdem kann die Schemaextraktion bereits vorhandene Schemata auf die für Partition-Miner und Integrator relevanten Strukturen reduzieren. In diesem Sinne erfüllt der Dienst Schemaextraktion eine komplexitätsreduzierende und vorverarbeitende, aber auch eine strukturerschließende Funktion.

Die Grundidee unseres Modells ist, die Struktur einer Menge von XML-Dokumenten mit Hilfe eines gerichteten Graphen zu beschreiben, der verschiedene Elementtypen zueinander in Beziehung setzt. Die gleiche Idee liegt auch anderen Modellen zugrunde, z. B. den sogenannten DataGuides,

die zur strukturellen Zusammenfassung von semistrukturierten Datenbanken entwickelt wurden [GW97]. Abb. 3.3 zeigt ein Beispiel des verwendeten Modells. Es wird ein kleines XML-Dokument und der extrahierte Schemagraph dargestellt.




```
<?xml version="1.0"?>

<root>
   <element>
      <child1>
         <common><element/></common>
      </child1>
      <ns:child2 xmlns:ns="namespace">
         <common/>
      </ ns:child2 >
   </element>
   <empty/>
   <mixed>
      Dies ist <em>Mixed</em> Content.
   </mixed>
</root>
```

Abb. 3.3: Einfacher Schemagraph zu einem XML-Dokument

4. Dokumentenverwaltung und inhaltsbasierte Suche

4.1 Dokumentenverwaltung

Nachfolgend gehen wir auf unsere Arbeiten zum zweiten wichtigen Aspekt der Dokumentenverwaltung ein. Dabei soll vor allem durch Einsatz von inhaltsbasierter und struktureller Suche die Qualität der Suchergebnisse wesentlich verbessert werden.

Die Suche nach Informationen im Internet liefert immer wieder Suchergebnisse von extrem schlechter Qualität, weil die heterogenen Dokumente weder systematisch nach ihrem Inhalt erschlossen wurden und kaum durch geeignete Metadaten beschrieben sind, noch die Suchmaschinen effektive Verfahren anwenden. Sie benutzen meist nur Schlüsselwort-basierte Suche und setzen vor allem keine Domänen-spezifischen Ontologien oder Klassifikationsschemata ein, um beispielsweise semantische Ähnlichkeit in den Griff zu bekommen. Um wesentlich bessere Suchergebnisse zu erzielen, muss zunächst mehr Aufwand in die inhaltliche Erschließung der Dokumente gesteckt werden, damit später dann Verfahren der Ähnlichkeitssuche (nach inhaltlichen und strukturellen Kriterien) gewinnbringend eingesetzt werden können. Deshalb betrachten wir das Problem der Dokumentensuche in einem größeren Zusammenhang und wollen uns dabei wiederum auf die Aspekte der Datenverwaltung konzentrieren.

4.1.1 Datenmodell

Es sind, wie bereits im vorangegangenen Bericht erläutert, folgende verschiedenen Arten von Daten auf jeweils spezifische Weise zu verwalten. Beschreibende Daten (auch Metadaten genannt)

folgen einem speziell zu entwickelnden Metadatenmodell. Ein Metadatensatz wird genau einem Dokument zugeordnet und die enthaltenen Attributwerte sind das Ergebnis der Erschließung dieses Dokuments. Neben beschreibenden Daten müssen die zugehörigen Dokumente selbst repräsentiert werden. Dies kann einerseits in Form von Referenzen (URLs) geschehen; es kann aber auch zusätzlich eine direkte Erfassung von Dokumenten mit anschließender Verwaltung von Quelldokumenten durch das Datenverwaltungssystem vorgesehen werden. Als dritte Gruppe von Daten sind die Personen- bzw. Organisations-bezogenen Daten zu nennen. Dies sind im wesentlichen Daten zu den verschiedenen Benutzergruppen, ihren Rollen (Autoren, Gutachter, Lernende usw.), Rechten/Pflichten und ihrer Einbindung in gewisse Organisationsstrukturen (Arbeitsgruppen, Fachbereiche, Institute usw.). Ebenfalls wichtig sind die ablaufbezogenen Daten, die Instanzen und Zustände von mehrschrittigen Abläufen („Workflows“) enthalten, sofern die Schemata dieser Abläufe an der Systemschnittstelle sichtbar sind und den Benutzern ihre Einbindung in solche Vorgänge klar ist.

Der Umfang an Attributen zur Beschreibung der Metadaten wurde für den Einsatz von XML in eine entsprechende formale Beschreibung der Struktur überführt. Die Verwendung von XML legt die Spezifikation der Struktur durch eine Document Type Definition (DTD[3]) oder durch ein XML-Schema-Dokument[4] nahe. XML-Schema erlaubt eine sehr differenzierte Spezifikation der Struktur und der verwendeten Datentypen innerhalb eines XML-Dokuments. Während eine DTD nur sehr einfache Beschreibungskonzepte anbietet, erlaubt XML-Schema beispielsweise den Einsatz von Abstraktionskonzepten wie die Spezialisierung (Vererbung). Da ein XML-Schema-Dokument allerdings ungleich schwieriger zu lesen ist als eine DTD und hier zunächst allein die Struktur und das Aussehen eines dieser Struktur entsprechenden XML-Dokuments wichtig ist, beschränken wir uns zunächst auf die Betrachtung von DTDs im Rahmen der Realisierung des prototypischen Systems. Im endgültigen System allerdings finden XML-Schema-Dokumente Einsatz.

Repräsentation der Metadaten durch XML

Es wurde bisher von Metadatensätzen und ihren Attributen gesprochen. Ein Metadatensatz beschreibt dabei eine Web-Ressource mit Hilfe einer Menge von Attributen, die in unserem Ansatz an den Dublin-Core-Standard[5] angelehnt sind. Das Attribut „Identifier“ beispielweise beinhaltet die URL des entsprechenden Lehr-/Lernangebots. Weitere Attribute wie „Creator“ oder „Format“ geben Auskunft über den Autor oder das verwendete Format. Ein diese Strukturen repräsentierendes XML-Dokument könnte wie in Abb. 4.1 aussehen.

Wie man in der Abbildung Abb. 4.1 auch leicht sehen kann, werden die Metadaten-Attribute auf XML-Elemente abgebildet. Angaben über den Ursprung („Origin“) des Inhalts eines XML-Elements bzw. Angaben zum Format oder Schema („Scheme“) werden unter Einsatz von XML-Attri-

3.) www.w3.org/xml
4.) www.w3.org/xml-schema
5.) www.dublincore.org

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Written by Marcus Flehmig "flehmig@informatik.uni-kl.de" -->
<metabase>
   <learningresource guid="K480000038">
      <title>
         <main origin="external">Vektoranalysis</main>
      </title>
      <creator origin="external">
         <name>Trautmann, Günther</name>
         <organization>Universität Kaiserslautern, Kaiserslautern, DE</organization>
      </creator>
      <subject scheme="freetext" origin="external" xml:lang="de">Vektoranalysis</subject>
      <date> <unspecified format="yyyy">1999</unspecified></date>
      <type>
         <document><vorlesungsskript/></document>
         <media><text/> </media>
      </type>
      <format>
         <mimetype>application/postscript</mimetype>
         <encoding>application/x-gzip</encoding>
      </format>
      <identifier href="http://www.mathematik.uni-kl.de/~wwwagag/preprints/skripte/Vektoranalysis.ps.gz"/>
      <language scheme="RFC1766">de</language>
      <educational>
         <verified>true</verified>
         <fieldofstudy><mathematik/></fieldofstudy>
      </educational>
      <annotations xml:lang="de">Kurzskript</annotations>
   </learningresource>
</metabase>
```

Abb. 4.1: Beispiel einer XML-Repräsentation

buten gemacht. Das mit XML verbundene semi-strukturierte Datenmodell erlaubt eine sehr intuitive Verwendung der Metadaten-Attribute. Bei der Erschließung eines Lehr-/Lernangebots können nicht in jedem Fall alle Metadaten-Attribute sinnvoll ausgefüllt werden, da es vorkommen kann, dass nicht alle Informationen verfügbar sind. Aber auch Mehrfachnennungen, bei denen nicht von Vorneherein die Kardinalitäten bekannt sind, kommen sehr häufig vor. So können beispielsweise beliebig viele Autoren („Creator“) an dem Verfassen eines Artikels beteiligt gewesen sein. Diese Freiheitsgrade können durch eine Strukturbeschreibung entsprechend sinnvoll eingeschränkt werden. Einen Auszug aus der verwendeten DTD findet man in Abbildung Abb. 4.2. Man erkennt, dass in der Datenbank („Metabase“) Lehr-/Lernangebote („Learning Resource“), Gutachten („Peer Review“) oder Benutzerkommentare („User Comment“) vorkommen können. Ein Lehr-/Lernangebote lässt sich vielfältig mittels weiterer XML-Elemente beschreiben. Dabei ist es z. B., wie man anhand der DTD sehen kann, notwendig, einen Titel anzugeben, aber es können beliebig viele Autoren genannt werden, was durch das Asterisk (*) hinter dem Elementnamen ausgedrückt wird. Mit Fragezeichen gekennzeichnete Elemente sind optional.

Jedes erfasste Lehr-/Lernangebote wird zunächst durch ein eigenes XML-Dokument repräsentiert. Allerdings können Lehr-/Lernangebote auch in Beziehung („Relation“) zueinander stehen. Die Definition der Metadaten-Attribute sieht verschiedene Arten von Beziehungen vor: Version, Teil

```
<!ENTITY % SubjectScheme "(SWD|DDC|MSC2000|PACS|RVK|freetext)">

<!ELEMENT   metabase (learningresource+, peerreview*, usercomment*)>
<!ELEMENT   learningresource (title, creator*, subject*, description?, publisher*,
            contributor*, date?, type?, format?, identifier*, source?, language*,
            relation?, coverage?, rights?, educational?, audience?, annotations?, image?)>
<!ATTLIST   learningresource guid ID #REQUIRED >
<!-- ============= Titel ==================================== -->
<!ELEMENT   title (main+, alternative*, version?)>
<!ELEMENT   main (#PCDATA)>
<!ATTLIST   main origin %OriginEnum; #REQUIRED
            xml:lang CDATA "en">
<!ELEMENT   alternative (#PCDATA)>
<!ATTLIST   alternative origin %OriginEnum; #REQUIRED
            xml:lang CDATA "en">
<!ELEMENT   version (#PCDATA)>
<!ATTLIST   version origin %OriginEnum; #REQUIRED
            xml:lang CDATA "en">
<!-- ============= Autoren ================================== -->
<!ELEMENT   creator (name, email*, organization*)>
<!ATTLIST   creator origin %OriginEnum; #REQUIRED>
<!ELEMENT   name (#PCDATA)>
<!ELEMENT   email (#PCDATA)>
<!ELEMENT   organization (#PCDATA)>
<!-- ============= Thema ==================================== -->
<!ELEMENT   subject (#PCDATA)>
<!ATTLIST   subject scheme %SubjectScheme; #REQUIRED
            origin %OriginEnum; #REQUIRED
            xml:lang CDATA "de">
```

Abb. 4.2: Auszug aus einer DTD

(„Part“), Format, sowie Verweise auf Gutachten („Review“) und Benutzerkommentare („User Comment“). Besteht beispielsweise ein Lehr-/Lernangebote aus verschiedenen Kapiteln, so können für jedes Kapitel eigene Einträge vorgesehen werden. Diese miteinander verknüpften Einträge werden durch ein einziges XML-Dokument repräsentiert, wobei die Verknüpfung durch XML-Referenzen realisiert wird. So können Strukturen (Beziehungen) sehr einfach gehandhabt und visualisiert werden, d. h. zusammengehörige Daten können gemeinsam durch nur eine Anfrage angezeigt werden. Auch das Erfassen von neuen Einträgen wird so optimal unterstützt. Es können zusammengehörige Daten gemeinsam ohne Kommunikationsaufwand mit der Datenverwaltungskomponente erfasst werden und als Ganzes zur weiteren Verarbeitung an die Datenbankkomponente geschickt werden. Um die Definition der XML-Dokumente bei der Erschließung weiter zu vereinfachen, können Referenzen auch an Ort und Stelle („inline“) definiert werden, d. h. ohne die Notwendigkeit, einen eigenen Eintrag für das Lehr-/Lernangebote zu erfassen. Allerdings geht damit die Möglichkeit verloren, zusätzlich einen Autor oder ähnliche Metadaten-Attribute für die referenzierten Einträge spezifizieren zu können. Aber so wird sehr einfach das Problem umgangen, lokal und Client-seitig bereits eindeutige Identifikatoren definieren zu müssen.

Ein Beispiel soll dies nun verdeutlichen: ein interessantes Skript aus dem Bereich Physik soll erschlossen werden. Es besteht aus mehreren Kapitel (hier drei) und das erste Kapitel enthält Verweise auf die Übrigen. Entsprechend obiger Ausführungen werden drei Einträge vom Type „Learning Resource“ definiert und miteinander verknüpft (siehe: Relation.HasPart.Reference-Element in Abb. 4.3). Müsste man bereits bei der initialen Erschließung Client-seitig die in Abb. 4.3 gezeigte Struktur erzeugen und an die Datenbankkomponente schicken, so ergäben sich eine Menge vermeidbarer Redundanzen und damit auch an eine Menge potenzieller Fehlerquellen. Daher ist für die initiale Erschließung eine Vereinfachung vorgesehen. In Abb. 4.4 findet man ein Beispiel für diese Vereinfachung, das die bereits genannte Inline-Definition verwendet. Bei dieser Variante werden in dem Relation.hasPart-Element des referenzierenden Eintrags nur die zusätzlichen Elemente definiert, die für die einzelnen Teile unterschiedlich sind.

Das Dokument aus Abb. 4.4 wird an die Datenbankkomponente geschickt und erst dort zu der Struktur aus Abb. 4.3 expandiert. Es befinden sich keine Inline-Definitionen in der Datenbank, so dass ein Pfadausdruck in einer Suchanfrage immer eindeutig ist. Die Werte des referenzierenden Eintrags werden übernommen. Bei Verwendung der Inline-Variante können ferner auch Angaben über das Format, den Dokument- bzw. Medien-Typ sowie auch zum Lehrmaterial gemacht werden. Die Möglichkeiten beschränken sich aber auf die Definition von Ein-Ebenen-Beziehungen, komplexe Strukturen lassen sich derart nicht definieren. Auch in dem Fall, dass jedem Eintrag beispielsweise ein eigener Autor zugewiesen werden soll, ist die Variante aus Abb. 4.3 zu wählen. Einen entsprechenden Auszug aus der DTD findet sich in Abb. 4.5.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<metabase>
   <learningresource guid="K480001276">
      <title><main origin="external">Cosmology: Mankind s Grand Investigation</main></title>
      <creator origin="external">
         <name>Bothun, Greg</name>
         <organization>University of Oregon, Eugene, US</organization>
      </creator>
      <subject scheme="freetext" origin="external" xml:lang="de">Kosmologie</subject>
      <subject scheme="freetext" origin="external" xml:lang="de">Weltall</subject>
      <subject scheme="freetext" origin="external" xml:lang="de">Außerirdisches Leben</subject>
      <identifier href="http://zebu.uoregon.edu/hb/c1w.html"/>
      <relation>
         <haspart>
            <inlineresource><identifier href="http://zebu.uoregon.edu/hb/c2w.html"/> </inlineresource>
         </haspart>
         <haspart>
            <inlineresource><identifier href="http://zebu.uoregon.edu/hb/c3w.html"/> </inlineresource>
         </haspart>
      </relation>
   </learningresource>
</metabase>
```

Abb. 4.4: Beispiel einer Inline-Definition

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<metabase>
      <learningresource guid="K480001276a">
      <title><main origin="external">Cosmology: Mankind s Grand Investigation</main></title>
      <creator origin="external">
         <name>Bothun, Greg</name>
         <organization>University of Oregon, Eugene, US</organization>
      </creator>
      <subject scheme="freetext" origin="external" xml:lang="de"> Kosmologie </subject>
      <subject scheme="freetext" origin="external" xml:lang="de"> Weltall </subject>
      <subject scheme="freetext" origin="external" xml:lang="de"> Außerirdisches Leben </subject>
      <identifier href="http://zebu.uoregon.edu/hb/c1w.html"/>
      <relation>
         <haspart><reference guid="K480001276b" /></haspart>
         <haspart><reference guid="K480001276c" / ></haspart>
      </relation>
   </learningresource>
   <learningresource guid="K480001276b">
      <title><main origin="external">Cosmology: Mankind s Grand Investigation</main></title>
      <creator origin="external">
         <name>Bothun, Greg</name>
         <organization>University of Oregon, Eugene, US</organization>
      </creator>
      <subject scheme="freetext" origin="external" xml:lang="de"> Kosmologie </subject>
      <subject scheme="freetext" origin="external" xml:lang="de"> Weltall </subject>
      <subject scheme="freetext" origin="external" xml:lang="de"> Außerirdisches Leben </subject>
      <identifier href="http://zebu.uoregon.edu/hb/c2w.html"/>
   </learningresource>
   <learningresource guid="K480001276c">
      <title><main origin="external">Cosmology: Mankind s Grand Investigation</main></title>
      <creator origin="external">
         <name>Bothun, Greg</name>
         <organization>University of Oregon, Eugene, US</organization>
      </creator>
      <subject scheme="freetext" origin="external" xml:lang="de"> Kosmologie </subject>
      <subject scheme="freetext" origin="external" xml:lang="de"> Weltall </subject>
      <subject scheme="freetext" origin="external" xml:lang="de"> Außerirdisches Leben </subject>
      <identifier href="http://zebu.uoregon.edu/hb/c3w.html"/>
   </learningresource>
</metabase>
```

Abb. 4.3: Beispiel einer Relation.HasPart-Beziehung

```
<!-- ============= Verweise ========================================== -->
<!ELEMENT relation (hasVersion|hasPart|hasFormat|hasReview|hasUserComments)+ >
<!ELEMENT hasVersion (reference|inlineresource) >
<!ELEMENT hasPart (reference|inlineresource) >
<!ELEMENT hasFormat (reference|inlineresource) >
<!ELEMENT hasReview (reference) >
<!ELEMENT hasUserComments (reference) >
<!ELEMENT reference EMPTY >
<!ATTLIST    reference guid IDREF #REQUIRED>
<!ELEMENT inlineresource (title?, type?, format, identifier, educational?, annotations?)>
```

Abb. 4.5: Strukturbeschreibung (DTD) der Inline-Definition

4.1.2 Verarbeitungsmodell

In der bisherigen Diskussion wurde immer wieder von XML-Elementen und XML-Dokumenten gesprochen. In Bezug darauf sind zwei verschiedenartige Verarbeitungsmodelle vorstellbar:

- ein Element-basiertes und
- ein Dokument-basiertes Verarbeitungsmodell.

Die Element-basierte Sichtweise betrachtet ein Element (also z. B. Autor oder Titel) als Granulat der Verarbeitung. Eine definierte Menge von Elementen bildet einen Metadatensatz und der Zustand der Erschließung ergibt sich allein aus der Menge der belegten Elemente. Leere Elemente können als Symbol für eine vorgenommene Eintragung dienen. Jedes Element kann für sich verwaltet, verändert und auch gesperrt werden. Auch die Autorisierung geschieht auf der Ebene der Elemente. Automatisch ausgefüllte bzw. gesammelte Elemente müssen alle einzeln verifiziert werden und für eine weitere externe Benutzung freigegeben werden. Diese feingranulare Verarbeitung ist aber in der Realisierung sehr aufwändig, teuer und in Betracht der zu realisierenden Anwendungen nicht sinnvoll. Aufgrund der eher einfachen Struktur der zugrundeliegenden Prozesse und damit vor allem aufgrund des grundlegenden MAS-Mehrschrittvorgangs erscheint ein gröberes Granulat der Verarbeitung sinnvoll. Wie es sich bereits im vorangegangen Abschnitt abgezeichnet hat, sind XML-Dokumente mit der bereits beschriebenen Struktur als Verarbeitungseinheit zu präferieren. Dokumente tragen dabei allerdings keine Verarbeitungsanweisungen oder -informationen, sondern repräsentieren allein die operationalen Daten. Die jeweilige Operation wird durch die verwendete WebDAV-Methode (Web-based Distributed Authoring and Versioning[6]) bestimmt, wobei die im vorangegangen Abschnitt beschriebenen Dokumente Parameter dieser Methoden sein können. Es können beispielsweise mittels WebDAV Sperren für ein Dokument gesetzt werden und so konkurrierende Zugriffe behandelt werden. Änderungen am Inhalt eines Dokuments können durchgeführt werden und nach der Bearbeitung kann das Dokument wieder an die Datenverwaltungskomponente zurückgeschickt werden. Dort würde das Dokument von einem WebDAV-Server verarbeitet und die enthaltenen Daten können in einem objekt-relationalen Datenbanksystem gespeichert werden.

Dokumenten-basierte Verarbeitung mit WebDAV

Die HTTP-Erweiterung WebDAV als Protokoll bzw. Schnittstelle zu der Datenverwaltungskomponente einzusetzen, besitzt den Vorteil, schon sehr früh mit der Entwicklung weiterer Komponenten (z. B. Redaktionssystem) beginnen zu können, die auf der Funktionalität der Datenverwaltungskomponente aufbauen, ohne dass diese bereits vollständig realisiert sein muss. Neben der Schnittstelle für die Definition der möglichen Operationen, ist mit der Beschreibung der Abbildung der Metadaten-Attribute auf XML ein konkretes Datenmodell gegeben, das dem Austausch der zu ver-

6.) www.webdav.org

waltenden Daten dienen kann. Um nicht einen vollständigen WebDAV-Server realisieren zu müssen, wurde ein Framework ausgewählt, das an den notwendigen Stellen erweitert werden kann.WebDAV kann auch vereinfacht als Web-basiertes Dateisystem bezeichnet werden, das neben HTTP-basierten Leseoperationen auch Schreiboperationen und den Einsatz von Sperren auf der Ebene von Dokumenten erlaubt.

4.2 WebDAV-basierte prototypische Realisierung mit Slide

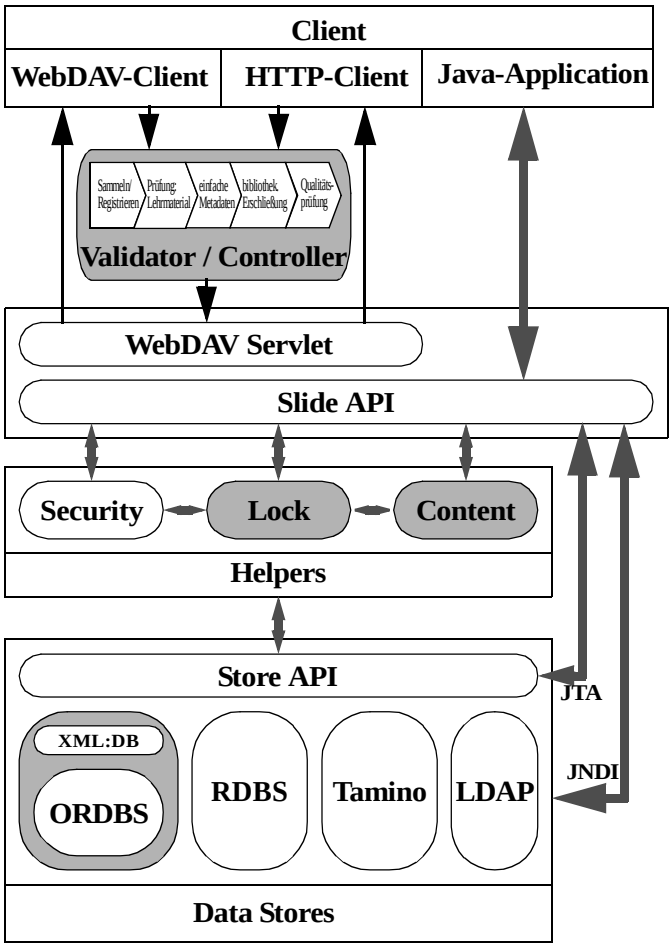


Abb. 4.6: Slide/WebDAV-basierter Architekturentwurf

Eine Realisierung des WebDAV-Standards stellt das Slide-Projekt[7] der Apache-Jakarta-Gruppe[8] dar. Slide besteht aus mehreren Server- und Client-Komponenten, zu deren Integration WebDAV verwendet und als genanntes Framework eingesetzt wird. Dieses Framework wird um eine spezialisierte Web-Anbindung zur Realisierung der Ablaufkoordination, um spezielle Helper-Komponenten für die Bereitstellung der Metadaten in XML sowie einer OR-basierten Data-Store-Implementierung ergänzt.

Eine Übersicht über die zugrundeliegende Systemarchitektur findet man in Abb. 4.6. Die grau hinterlegten Komponenten markieren die Stelle, an denen eine Anpassung notwendig ist. Neben der Erweiterung durch die Realisierung zusätzlicher Komponenten kommt aber bei dem Einsatz eines Frameworks auch der Konfiguration (hier: das WebDAV-basierte Content-Management-System Slide) eine wichtige Bedeutung zu. Zu diesem Zweck wurden zunächst die verschiedenen Rollen identifiziert, deren möglichen Operationen in den verschiedenen Stufen des Erschließungsprozesses bestimmt und die Rechte entsprechend spezifiziert. Dies geschah unter Berücksichtigung der besonderen Eigenschaften des WebDAV-Protokolls bzw. unter Berücksichtigung der Eigenarten Webbasierter Anwendungen. Für die Abbildung der ver-

7.) jakarta.apache.org/slide
8.) jakarta.apache.org

schiedenen Verarbeitszustände eines Dokuments wurde naheliegenderweise für jeden Zustand ein eigenes Verzeichnisse erzeugt. Dokumente eines bestimmten Verarbeitungszustandes befinden sich somit in den jeweiligen Verzeichnissen und auf sie kann so sehr leicht mittels WebDAV zugegriffen werden. Zusätzlich können über so genannte Properties detailliertere Informationen über den Zustand der Erschließung mittels WebDAV angefragt werden. Auch darüber hinausgehende Informationen, beispielsweise über das betreffende Studienfach (Physik, Biologie etc.) können über Properties bezogen werden, so dass die Erschließung optimal unterstützt werden kann. Um die Spezifikation der Rechte bestimmte Operationen ausführen zu dürfen, müssen zunächst die möglichen Rollen identifiziert werden, die ein Benutzer im System einnehmen kann.

Rollen

Rollen ermöglichen es, die Interaktionen zwischen den Teilnehmern zu strukturieren und die Funktionalitäten abhängig von der Rollenverteilung zu definieren. Zwei Aspekte sind zu berücksichtigen [BS95]:

1. Die Rolle definiert die soziale Funktion eines Einzelnen in Beziehung zum Gruppenprozess, zur Organisation und zu anderen Gruppenteilnehmern.
2. Die Rolle definiert die Rechte und Pflichten im Rahmen des Gruppenprozesses. Die Kontrolle über die Informationseinheiten (z. B. Lese- und Schreibrechte) und die Aktivitäten, welche die Einzelnen ausführen dürfen oder müssen, werden festgelegt. Ebenso können Privilegien vergeben werden.

Die Informationseinheiten im o. g. Sinn bestehen aus Dokumenten, d. h., das Granulat der Verarbeitung ist das Dokument. Diese Dokumente sind XML-basiert und somit textbasiert. Es gibt Dokumente, die Metadaten, Gutachten, Benutzerkommentare oder Kombinationen beinhalten können. Die Schnittstelle für den Web-basierten bzw. entfernten Zugriff bildet WebDAV. Entsprechend der o. g. Definition lassen sich aus den Erschließungsstufen direkt die primären Rollen ableiten:

- Endbenutzer (auch externer Benutzer bzw. WebUser),
- Redakteur,
- Gutachter,
- Systemverwalter.

Durch die Trennung von formaler und inhaltlicher Erschließung wird allerdings eine Verfeinerung der Rolle Redakteur notwendig:

- Zuarbeiter (wissenschaftliche Hilfskräfte oder automatische Systeme),
- Bibliothekar,
- Verantwortlicher für den Inhalt (Administrator).

Ferner ist eine besondere Rolle für die Verwaltung (Management) des Systems sinnvoll. Dieser so genannter Maintainer ist nicht verantwortlich für den Inhalt im Sinne des Administrators, sondern

für die Verwaltungsaufgaben innerhalb des Systems. Die möglichen Aufgaben betreffen beispielsweise die Benutzerverwaltung oder Datensicherung. Somit wurde die ursprüngliche Systemverwalterrolle in eine technische und eine nicht-technische Rolle unterteilt.

Zusammenfassend lassen sich die Rollen wie folgt definieren:

- WebUser: Endbenutzer mit Suchmöglichkeiten (lesender Zugriff) und der Möglichkeit, Link-Vorschläge und Kommentare in die Sammlung einzubringen.
- Bibliothekar: Die Rolle der Bibliothekare umfasst alle Kompetenzstufen bis auf den Gutachter, d. h., ein Bibliothekar kann auf allen Stufen des Erschließungsprozesses wirken und auch nur ein Bibliothekar kann entscheiden, ob ein XML-Dokument mit den zugehörigen Metadaten öffentlich zugänglich gemacht werden soll.
- Robot: Ein Robot beschreibt die Rolle eines Zuarbeiters, die einem Werkzeug zum automatischen Sammeln und Erschließen zugewiesen wird. Solche Helfer können allerdings nur intern wirken und zuarbeiten. Ein Bibliothekar muss von ihnen erfasste Daten validieren und explizit einer Verwendung zustimmen.
- HiWi: Diese Rolle beschreibt wissenschaftliche Hilfskräfte. Auch solche Helfer können nur intern wirken und zuarbeiten. Auch die von ihnen erfassten Daten müssen validiert werden und es muss explizit einer Verwendung zugestimmt werden.
- Gutachter: Die genaue Funktion der Rolle des Gutachters muss im Detail noch definiert werden. Es ist aber schon hier wichtig zu erwähnen, dass Benutzer in dieser Rollen nur auf ihre eigenen Gutachten uneingeschränkt zugreifen können dürfen. Es sollte beispielsweise nicht möglich sein, dass ein Gutachter Gutachten Anderer ändern kann.
- Admin: Die Administratorrolle ist für die Inhalte des Autorensystems verantwortlich und kann Änderungen unabhängig von den verschiedenen Verarbeitungsstati durchführen. Sie repräsentiert letztlich einen Bibliothekar mit umfassenden Rechten in Bezug auf die jeweiligen Inhalte bzw. Dokumente.
- Maintain: Die Funktion der Systemadministration und Wartung kommt der Maintainer-Rolle zu. Benutzern ist es erlaubt, in dieser Rolle auch so genannte Managementfunktionen durchzuführen (bspw. Benutzerverwaltung, Sicherungen etc.).

4.3 Produktionssystem

Die HTTP-Erweiterung WebDAV als Protokoll bzw. Schnittstelle zu der Datenverwaltungskomponente einzusetzen, hatte den Vorteil, schon sehr früh mit der Entwicklung weiterer Komponenten beispielsweise dem Redaktionssystem beginnen zu können, ohne dass eine vollständige Implementierung des Gesamtsystems hätte existiert müssen. Neben der Schnittstelle für die Definition der möglichen Operationen ist mit der Beschreibung der Abbildung der Metadaten-Attribute auf XML ein konkretes Datenmodell gegeben, das dem Austausch der zu verwaltenden Daten dienen kann.

Die prototypische Realisierung der Datenverwaltungskomponente mittels des WebDAV-Standards durch den Einsatz des Slide-Projekts der Apache-Jakarta-Gruppe erlaubte innerhalb kurzer Zeit, ein evaluierbares System zur Verfügung zu haben. Für die Realisierung der verfeinerten Konzepte allerdings, reichte die Erweiterbarkeit des Frameworks nicht aus. Daher wurde die Slide-basierte Architektur im Kern, d. h. an den Stellen der Dokumenten- bzw. Datenverwaltung, durch eine eigene Realisierung ersetzt. Eine Anpassung der Slide-spezifischen Komponenten hätte teilweise einen größeren Aufwand dargestellt als eine eigene Implementierung. Dies lag insbesondere daran, dass sich die Anforderungen in Bezug auf die Web-basierte Suche, dem Redaktionssystem und dem zugrundeliegenden Erschließungsprozess und damit auch an die Datenverwaltungskomponente zu einer sehr späten Phase des Projekts nicht unwesentlich geändert hatten. Die erweiterte Implementierung basiert auf dem Java™ 2-Enterprise-Edition-Framework (J2EE)[9] und nutzt dessen Vorteile durch Einsatz von Enterprise JavaBeans™ und anderen wohl etablierten Techniken. Dabei soll auch weiterhin WebDAV als eine der möglichen Schnittstelle unterstützt werden.

4.3.1 Java2-Enterprise-Edition: Plattform für unternehmensweite Anwendungen

Die Java™ 2 Platform, Enterprise Edition (J2EE) bildet den Standard für unternehmensweite Java-basierte Anwendungen. Durch die Definition von allgemeinen Diensten (beispielsweise Transaktionsverwaltung, Nachrichten-, Namens- oder Verzeichnisdiensten) und die systemseitige Unterstützung fundamentaler Anwendungssemantik (beispielsweise Persistenz von Geschäftsobjekten) lassen sich nun mehrschichtige Architekturen definieren, die allein auf standardisierten, modularen Komponenten basieren. Dabei baut J2EE auf etablierten Techniken und Konzepten der Java2-Standard-Edition (J2SE) auf, geht mit der Unterstützung für JDBC™ (Datenbankzugriff), CORBA (Kommunikation in verteilten, heterogenen Systemumgebungen), Enterprise-JavaBeans™-Komponenten (EJB), Java-Servlet-Schnittstelle, Java-Server-Pages™ (JSP) und XML weit darüber hinaus und überträgt das mit Java im Allgemeinen verbundene Konzept „Write Once, Run Anywhere™“ auf unternehmensweite Anwendungen: „With simplicity, portability, scalability and legacy integration, J2EE is the platform for enterprise solutions.“ (SUN Microsystems, 2000).

4.3.2 J2EE-Programmier- und -Anwendungsmodell

Dem durch J2EE definierten Modell für die Entwicklung unternehmensweiter Anwendungen liegt die Trennung von Präsentations-, Anwendungs- und Datenhaltungsaspekten zugrunde. Die Trennung erfolgt durch die Definition unterschiedlicher Schichten für die Realisierung der Präsentationslogik, Anwendungslogik („Server-Side Business Logic“) und Datenhaltungskomponente („Enterprise Information System“). Die Präsentationsschicht ist ihrerseits wieder unterteilt in eine Client-seitige und eine Server-seitige Schicht. In der Client-seitigen Präsentationsschicht sind ver-

9.) java.sun.org/j2ee

schiedene Endgeräte berücksichtigt. So ist vorgesehen, über Web-Browser („HTTP-User-Agents“), Applets oder auch eigenständigen Java-Clients auf die entsprechenden Server-seitigen Komponenten zugreifen zu können. Die Server-seitige Präsentationsschicht wird ausgefüllt durch den Web-Server. Dieser beinhaltet einen so genannten Web-Container zur Ausführung von Servlets und JSP-Seiten. Den EJB-Container zur Ausführung von EJB-Komponenten findet man in der Schicht der Server-Side-Business-Logic. Web-Server bzw. Web-Container und EJB-Container bilden zusammen mit den entsprechenden Diensten den so genannten J2EE-Application-Server. Datenbanksysteme, bzw. so genannte Enterprise-Information- Systems (EIS), findet man auf der Ebene der Datenhaltung wieder.

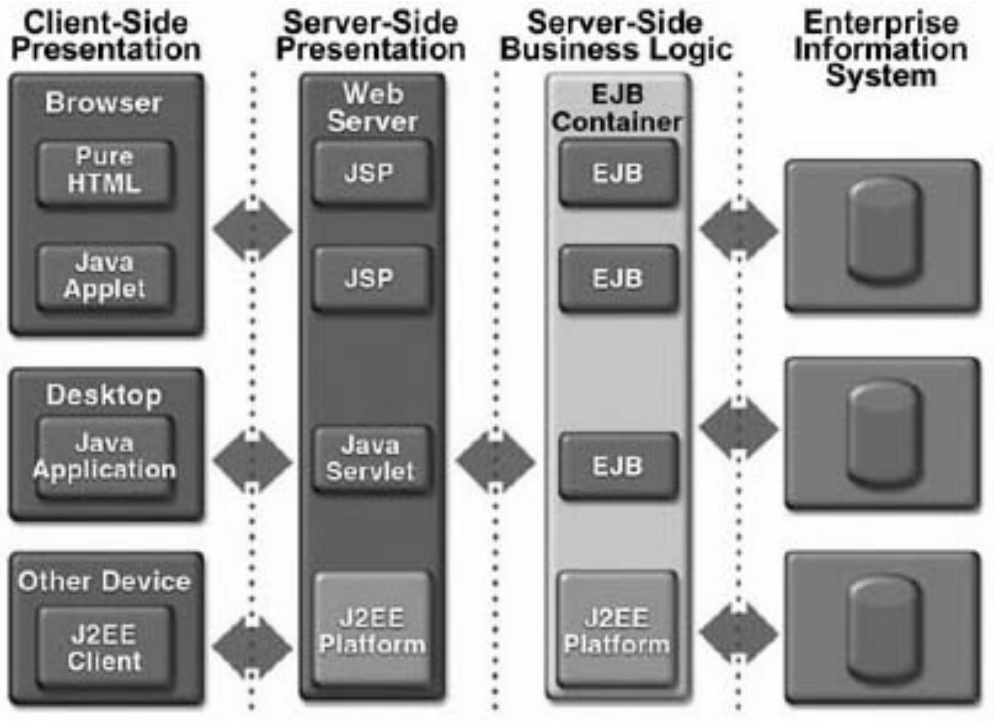


Abb. 4.7: J2EE Mehrschichtenarchitektur

4.3.3 Enterprise-JavaBeans-Komponenten

Enterprise-JavaBeans (EJB) sind Komponenten eines flexiblen Komponentenmodells für Geschäftsobjekte in verteilten Systemumgebungen. Ein Komponentenmodell ermöglicht die Erstellung von wieder verwendbaren Software-Teilen (Komponenten). Es beschreibt eine Infrastruktur für den Einsatz und die Kommunikation von diesen so genannten Komponenten. Eine Komponente ist ein Stück Software, das klein genug ist, um es in einem Stück erzeugen und pflegen zu können, groß genug ist, um eine sinnvoll einsetzbare Funktionalität zu bieten und eine individuelle Unterstützung zu rechtfertigen sowie mit standardisierten Schnittstellen ausgestattet ist, um mit anderen Komponenten zusammenzuarbeiten [Gri98]. Komponenten und damit auch EJBs können zur Installationszeit konfiguriert und angepasst werden. Ihr Verhalten kann bedingt deklarativ spezifiziert

werden. J2EE kennt zwei unterschiedliche Arten von EJB-Komponenten: Session-Beans und Entity-Beans. Entity-Beans repräsentieren persistente Geschäftsobjekte (beispielsweise im einfachsten Fall ein einzelnes Tupel in einer Tabelle einer relationalen DB), während Session-Beans das Verhalten bzw. die Geschäftslogik realisieren und somit auch das Zusammenspiel von Entity-Beans organisieren. Die Erwartungen, die in J2EE gesetzt werden, sind durch vollmundige Ankündigungen entsprechend hoch, es bleibt allerdings abzuwarten, ob diese Erwartungen auch erfüllt werden können: “Based on these flexible component configurations, the J2EE application model means quicker development, easier customization and greater ability to develop powerful enterprise applications. And, because it is based on the Java programming language, this model enables all J2EE applications to achieve all the benefits of Java technology: scalability, portability, and programming ease.“ (Sun Microsystems, 2002).

4.3.4 Übersicht über den J2EE-basierten Implementierungsansatz

Um den gewachsenen Ansprüchen an die Datenverwaltungskomponente gerecht zu werden, wurde bei der Erweiterung des bisherigen Ansatzes auf sinnvolle Modularität geachtet. Die Re-Implementierung mittels Java 2 Enterprise-Edition (J2EE) umfasst daher folgende Teilbereiche (siehe Abb. 4.8):

- Anfrageverarbeitung (Query-Engine),
- Ergebnisverarbeitung (Result-Set-Processor),
- XML-Dokumenten-Management (XML-Processor),
- Web-basierte Suche (Web-Search),
- Redaktionssystem (Indexing-Tool) und
- Support-Komponenten (bspw. zur Unterstützung der Navigation über die RVK-Klassifikation oder Benutzerverwaltung).

Dabei konnten die Ansätze aus den bisherigen Arbeiten aufgrund entsprechender Kapselung und Festhalten an Java ohne große Änderungen weiterverfolgt werden. Alle genannten Teilbereiche sind durch entsprechende Diplom- oder Projektarbeiten bzw. Studenten (d. h. Hiwis) besetzt. Die Web-basierte Suche und die entsprechende Infrastruktur zum Testen der Such-Schnittstelle sind bereits nahezu vollständig implementiert [FFW02]. Diese Testumgebung wird nun sukzessive durch die verschiedenen Projekt- und Diplomarbeiten mit Funktionalität ausgefüllt. Mit der Entwicklung des Redaktionssystems basierend auf der neuen Technologie wurde bereits begonnen.

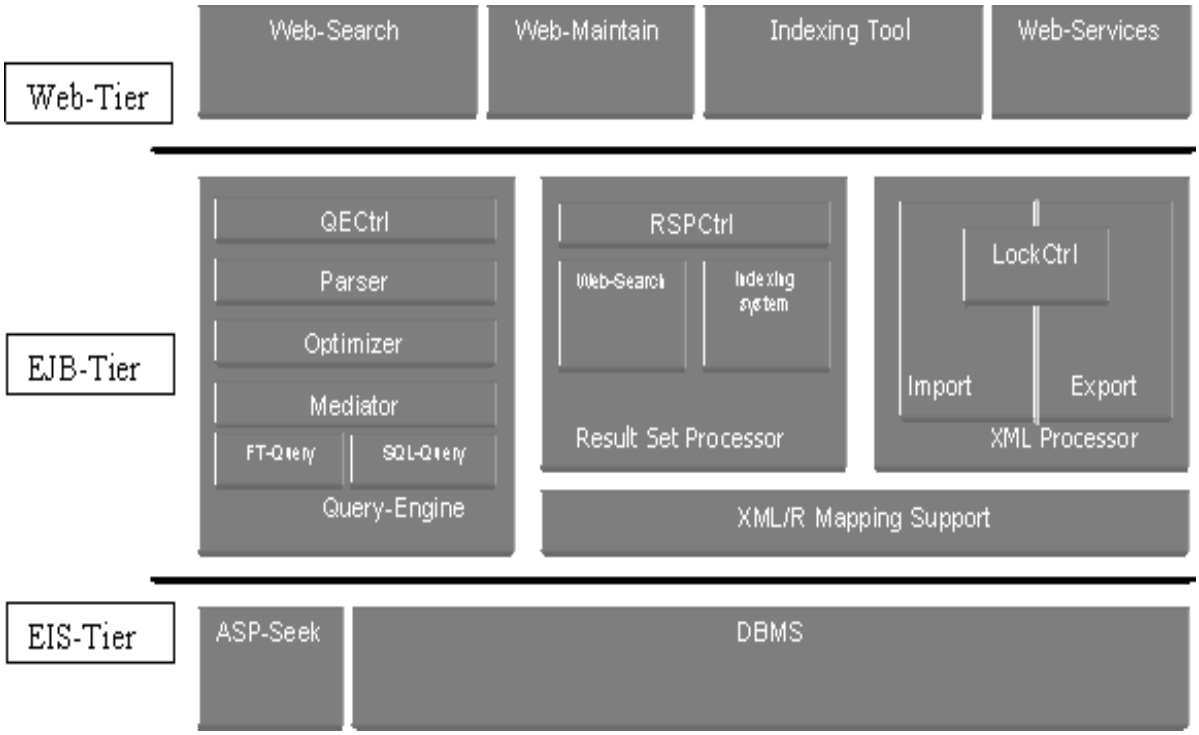


Abb. 4.8: Der J2EE-basierte Implementierungsansatz

Grundlegende Konzepte der Re-Implementierung

Der Abbildung deutet ferner an, wie die genannten Komponenten in Beziehung stehen. Die Datenhaltungsschicht („EIS-Tier“), bestehend aus einem relationalen Datenbanksystem und einem Volltextindexierungssystem, bildet die Basis. Auf dieser baut die mittlere Schicht der EJB-Komponenten auf („EJB-Tier“). EJBs realisieren die eigentliche Anwendungslogik. Die Schicht der Serverseitigen Präsentation („Web-Tier“) wiederum nutzt die EJB-Komponenten und realisiert die verschiedenen Module beispielsweise zur Unterstützung der Web-basierten Suche, der Redaktionsschnittstelle oder so genannter Web-Services.

Nochmals soll besonders betont werden, dass die Aufgaben der eigentlichen Anfrageverarbeitung („Query-Engine“), der Ergebnisverarbeitung (Result-Set-Processor) und des XML-Dokumenten-Managements („XML-Processor“) getrennt betrachtet werden und auch separat realisiert werden („Separation of Concerns“). Es soll nun auf die Komponenten im Einzelnen eingegangen und ihre Besonderheiten näher erläutert werden.

Basisfunktionalität (Foundation)

Zunächst seien die fundamentalen Dienste erwähnt, die eine besondere Unterstützung der Regensburger Verbund Klassifikation (RVK) realisieren, sowie Unterstützung für Schema-Informationen (bspw. Wertebereiche, vordefinierte Vokabularien oder interne Abbildungsinformationen), Benutzerverwaltung und Konfiguration anbieten. Die Aufgabe der Verwaltung der strukturierten Daten

übernimmt ein objekt-relationales Datenbankverwaltungssystem (ORDBVS). Die Verwaltung von unstrukturierten Daten wird durch ein entsprechendes Volltext-System (hier: ASP-Seek) unterstützt und bei der Anfrageverarbeitung integriert.

XML-Dokumenten-Management (XML-Processing)

Die jeweiligen Metadatensätze und die miteinander verknüpften Einträge werden, wie eingangs bereits erwähnt, durch ein einziges XML-Dokument repräsentiert, wobei die Verknüpfung durch XML-Referenzen realisiert wird. So können Strukturen (Beziehungen) sehr einfach gehandhabt und visualisiert werden, d.h., zusammengehörige Daten können gemeinsam durch nur eine Anfrage angezeigt werden. Auch das Erfassen von neuen Einträgen wird so unterstützt. Es können zusammengehörige Daten gemeinsam erfasst und als Ganzes zwecks weiterer Verarbeitung verschickt werden. Da die Verarbeitung also Dokumenten-orientiert erfolgt, muss die XML-Processor-Komponente die Aufgabe übernehmen können, XML-Dokumente auszutauschen. Diese Import- und Exportfunktionalität von Meta-Akad-Dokumenten wird durch geeignete Sperrmechanismen weiter unterstützt.

Anfrageverarbeitung (Query-Engine)

Anfragen an die Menge der Meta-Akad-Dokumente sollten beliebig formuliert werden können, d. h., es sollten beliebige Meta-Akad-Attribute bzw. XML-Elemente benutzt werden können, um einfache Prädikate zu bilden und aus diesen dann komplexe Ausdrücke zusammenzusetzen. Ferner sollte eine Volltextsuche unterstützt werden und es sollte möglich sein, statistische Informationen („Meta-Queries“) über das Anfrageergebnis zu erfahren, beispielsweise über die Anzahl der Treffer, die häufigsten Schlagworte oder Klassifikationen. Es sollte aber auch möglich sein, das Anfrageergebnis nach möglichst beliebigen Kriterien zu sortieren. Möchte man nun aber die Anfrage nicht mehrfach ausführen müssen, da die notwendigen Schritte gegebenenfalls sehr kostenintensiv sein können, so liegt es nahe, die Anfrageergebnisse vollständig zu materialisieren. Die Materialisierung erlaubt auch eine sinnvolle Trennung von Anfrageverarbeitung und Ergebnisverarbeitung.

Die Anfrageverarbeitung umfasst im Allgemeinen die syntaktische Analyse der Anfrage, ihre Optimierung, Übersetzung und Ausführung. Da aber im Falle der Anfrageverarbeitung im Meta-Akad-Projekt zwei Datenquellen mit unterschiedlichem Datenmodell zu kombinieren sind, wurde ein zusätzlicher Verarbeitungsschritt notwendig. In diesem Schritt übernimmt ein so genannter Mediator die Zerlegung der Anfrage und die Kombination der Ergebnismengen.

Die Teilanfragen werden der jeweiligen Ausführungskomponente übergeben. Die Ausführungskomponente zur Unterstützung des Volltextindexierungssystems besitzt die Fähigkeit, Anfragen in einem Cache zwischenzuspeichern und wieder zu verwenden. Das integrierte Gesamtergebnis wird in der relationalen Datenbank materialisiert und kann von der Ergebnisverarbeitung weiterverwendet werden.

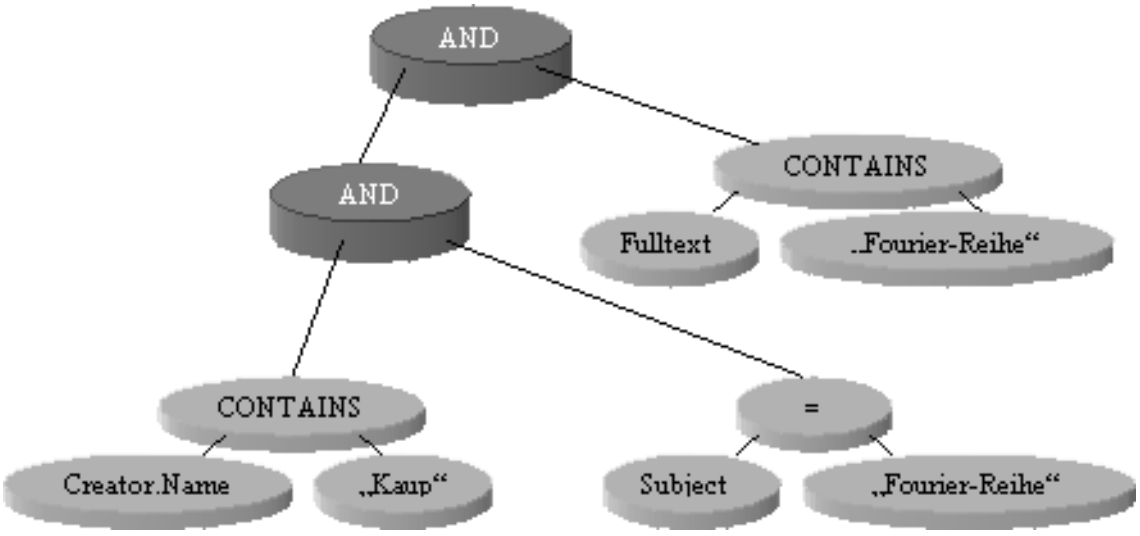


Abb. 4.9: Anfrageverarbeitung mittels eines Query-Tree

Die Anfrageverarbeitung führt somit die Anfrage nur in dem Sinne aus, dass sie die interne Repräsentation einer Anfrage („Query-Tree“, Abb. 4.9) in entsprechende SQL-Anfragen bzw. Volltextsuchanfragen transformiert und ausführt. Ergebnisse werden in temporären Tabellen gespeichert, was wiederum zu einem günstigeren Zugriffsverhalten auf der Datenbankebene führt, da die Sperrproblematik entschieden entschärft wird. Des Weiteren ermöglicht die Trennung überhaupt erst das Konzept des Stateless-Cursor, welches im nachfolgenden Abschnitt im Rahmen der Ergebnisverarbeitung beschrieben werden soll.

Ergebnisverarbeitung (Result-Set-Processing)

Das materialisierte Anfrageergebnis dient der Ergebnisverarbeitung als Grundlage für die Beantwortung von weiteren Anfragen nach der Anzahl von Treffern oder den am häufigsten verwendeten Schlagworten. Diese Anfragen und auch die Behandlung der Sortierung der Anfrageergebnisse nach den verschiedenen Kriterien erfolgt ohne die wiederholte Ausführung der gesamten Anfrage. Ebenso ist der Wechsel zwischen verschiedenen bereits schon einmal ausgewählten Sortierungen ohne erneute Ausführung möglich.

Dabei dient als Schnittstelle zum Web-Tier der Result-Set-Processor-Controller (RSPCtrl). Die RSPCtrl-Komponente unterstützt zwei unterschiedliche Implementierungen: eine zustandslose („stateless“) und eine zustandsbehaftete („stateful“) Variante. Zwischen diesen beiden kann frei gewählt werden. Allerdings ist die zustandslose für den Zugriff über das Web optimiert und benutzt das bereits genannte Stateless-Cursor-Konzept, während die andere Variante in Bezug auf Zugriffe durch das Redaktionssystem optimiert wurde. Letztere bietet weitere Sortiermöglichkeiten an und benutzt Datenbank-spezifische Cursor zum Durchlaufen der Ergebnismenge. Dies schont zwar die Datenbank-seitig aufzubringenden Ressourcen für die Speicherung der Ergebnisse, allerdings erhöht sich die Anzahl der offenen Datenbank-Verbindungen, die nicht gemeinsam benutzt werden können.

4.3.5 Stateless-Cursor-Konzept

Bei Zugriffen über die Web-basierte Suchschnittstelle müssen viele parallele Anfragen gleichzeitig verarbeitet werden können. Im Allgemeinen sollte daher ein Ziel immer die gemeinsame Nutzung von DB-Ressourcen sein. Eine besonders kritische Ressource stellt in der Regel die Datenbankverbindung dar. Daher wird bei der Web-basierten Suche versucht, die Dauer der exklusiven Nutzung einer Datenbankverbindung zu minimieren. Aus diesem Grund erhält jedes Objekt bzw. Tupel, das sich bei einer Anfrage qualifiziert hat, einen eindeutigen und fortlaufenden Schlüssel, über den Bereichsanfragen gestellt werden können. So ist es möglich, auf Teilergebnisse zuzugreifen, ohne während der Verarbeitung und insbesondere ihrer Benutzer-seitigen Präsentation eine Datenbankverbindung oder einen Datenbank-Cursor zu halten. Soll über die Gesamtmenge der Ergebnisse iteriert werden, können sukzessive Pakete beliebiger Größe angefordert werden. Die dabei zu verwaltende Cursor- bzw. Iterator-Position wird in der Datenbank abgelegt. Es muss nur der entsprechende Schlüsselwert des zuletzt gelesenen Objekts bzw. Tupels gespeichert werden. Um auch die Kosten für Auf- bzw. Abbau einer Verbindung zu minimieren, werden die Verbindungen in einem so genannten Pool verwaltet. Auch die Komponenten (d. h. Session-EJBs), welche die oben genannten Bereichsanfragen verarbeiten, werden gemeinsam verwendet und in einem Pool verwaltet, um die Kosten bei Instanzierung und Löschung zu minimieren. Diese insgesamt zustandlose Verarbeitung ist essentiell für das Stateless-Cursor-Konzept.

4.3.6 Beispielhafte Anfrageverarbeitung

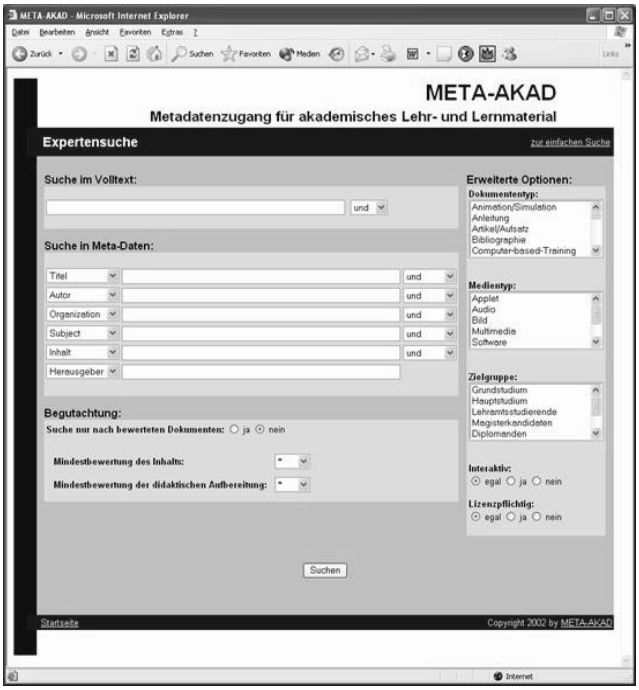


Abb. 4.10: Expertensuchschnittstelle

Auf die Anfrageverarbeitung soll nun anhand eines Beispiels näher eingegangen werden. Zunächst muss eine Query-Tree-Objekt-Struktur (siehe Abb. 4.9) definiert werden, welche die eigentliche Anfrage repräsentiert. Dieser so genannte Query-Tree wird beispielsweise von einer Komponente der Web-basierten Suche nach Ausfüllen des Formulars der erweiterten Suche generiert und dem Query-Engine-Controller (QECrl) übergeben, der als zentraler Einstiegspunkt der Query-Engine-Komponente fungiert (siehe Abb. 4.10). Die Query-Engine verarbeitet die Anfrage und erstellt, wie bereits erwähnt, eine temporäre Tabelle und lässt, wiederum durch die QECtrl, ein Stellvertreter-Objekt („Proxy“ oder „Handle“) für das Anfrageergebnis zurückliefern.

Dieses Handle wird nun bei jedem der weiteren Schritten als Kontextinformation benötigt und muss den weiteren Methodenaufrufen mitgegeben werden. Es muss daher zunächst auch dem Result-Set-Processor-Controller (RSPCtrl) übergeben werden, über den es möglich ist, eine Übersicht der sich qualifizierenden Tupel zu beziehen und über die gesamte Ergebnismenge zu iterieren (s.Abb. 1.5).

Anhand eines eindeutigen Identifikators kann von dem XML-Processor das vollständige Metadaten-Dokument bezogen werden und nach Anforderung einer Schreibsperre auch verändert werden.

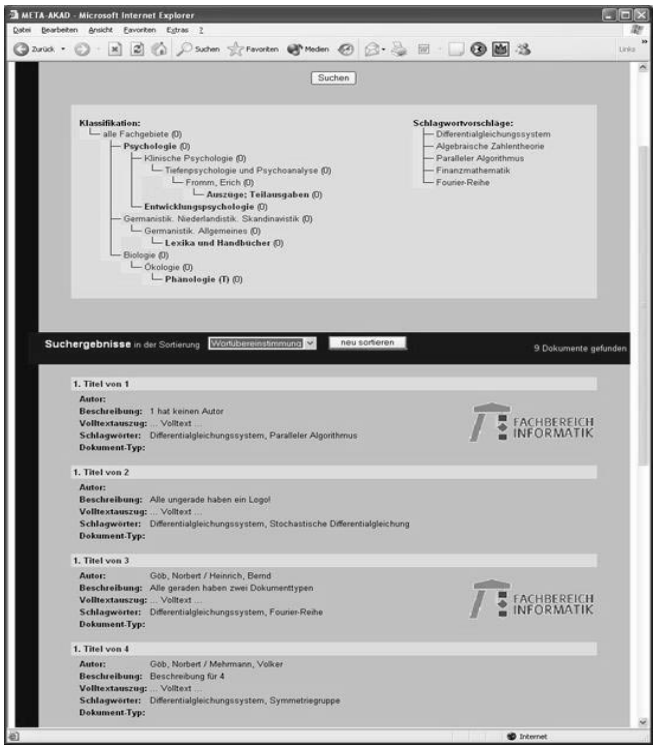


Abb. 4.11: Präsentation der Ergebnisliste

4.4 Realisierung der Anfrageverarbeitung

Dieser Abschnitt beschreibt die Anforderungen, den Ablauf, die Implementierung der Anfrageverarbeitung und die Funktionsweise einzelner Komponenten. Die Anfrageverarbeitung (AV) ist eine der drei Hauptkomponenten der Anwendungslogik der Dokumentenverwaltung. Sie hat zur Aufgabe, möglichst beliebige Suchanfragen entgegenzunehmen und ein entsprechendes Ergebnis zurückzuliefern. Es soll einfach möglich sein, aus den Metadaten-Attributen bzw. aus den zugehörigen XML-Elementen Prädikate (beispielsweise **Autor = 'Göb, Norbert'**) zu bilden und diese dann zu komplexen Ausdrücken zusammenzusetzen (z. B. **[Autor = 'Göb, Norbert'] AND [Titel enthält 'Algebra']**). Weiterhin sollte die Möglichkeit bestehen, die Suche in den strukturierten Meta-Daten mit einer Volltextsuche zu kombinieren. Auch muss die Möglichkeit berücksichtigt werden, Anfragen zu materialisieren und in einem Cache abzulegen, um kostenintensive Wiederholungen von Anfragen (beispielsweise bei Umsortierung, Anfrageverfeinerung und statistischer Auswertung über das gesamte Anfrageergebnis) effizienter bearbeiten zu können. Das von der AV gelieferte Ergebnis muss in einer Form vorliegen, die es erlaubt, die AV sinnvoll von der Ergebnisverarbeitung zu trennen. Es sollte möglich sein, das Ergebnis nach verschiedenen Kriterien zu sortieren und eine aktuelle Position (Offset des Cursor) zu setzen. Falls eine Volltextsuche durchgeführt wird, soll ein zu dem jeweiligen Dokument passender Volltextauszug mitgeliefert werden. Die Ergebnisse einer Anfrage werden daher in temporären Tabellen Datenbank-seitig gespeichert, was zu einem einfachen Zugriff seitens der Ergebnisverarbeitung führt.

Im Wesentlichen besteht die Aufgabe der AV in der syntaktischen Analyse der Anfrage, ihrer Optimierung, Übersetzung und Ausführung. Insbesondere soll jedoch eine Anfrage an zwei verschiedene Datenhaltungssysteme gestellt werden können, die sich zudem in ihrem Datenmodell stark unterscheiden: Zum einen ein relationale DBS mit Metadaten und zum anderen ein Volltextindexierungssystem. Die AV muss Anfragen an beide Systeme stellen und die Ergebnisse entsprechend kombinieren können.

4.4.1 Phasen der Anfragenverarbeitung

Die Verarbeitung einer Anfrage kann mit einem einfachen „Top-Down“-Ansatz beschrieben werden. An [Mit95] angelehnt kann die AV in fünf Phasen (siehe Abb. 4.12) eingeteilt werden:

Schritt 1: *Interndarstellung der Anfrage*. Um die Anfrage effizient verarbeiten zu können, wird sie vom *QueryParser* in ein geeignetes internes Format umgewandelt. Diese interne Darstellung soll einfach zu übersetzen sein und die anschließend folgende Transformation der Anfrage unterstützen. Weiterhin wird eine Syntaxanalyse der Anfrage durchgeführt.

Schritt 2: *Transformation der Anfrage*. Logische Umformungen sollen die Anfrage standardisieren und, falls erforderlich, auch zusammenfassen. Geeignete Transformationen führen zu einem effizienteren Anfrageplan.

Schritt 3: *Erzeugung eines Anfrageplans*. Der *Plangenerator* erzeugt aus der internen Darstellung einen Anfrageplan (QueryPlan), der die Grundlage für die Generierung der späteren SQL-Anfrage an die relationale Datenbank bildet. Ferner führt er Vorbereitungen für die Volltextsuche durch, indem er entsprechende temporäre Tabellen in der Datenbank erstellt.

Schritt 4: *Optimieren des Anfrageplans.* Der im letzten Schritt erzeugte Anfrageplan führt nicht automatisch zu einer optimalen SQL-Anfrage. Der *PlanOptimizer* versucht nun den Anfrageplan entsprechend umzuformen, um eine möglichst optimale SQL-Anfrage zu erhalten.

Schritt 5: *Ausführung des Anfrageplans*. Im fünften und letzten Schritt wird aus dem Anfrageplan eine SQL-Anfrage generiert und ausgeführt. Das Resultat der Anfrage wird entsprechend aufbereitet und in eine temporäre Tabelle geschrieben. Ein Proxy-Objekt dient im Weiteren als Stellvertreter für das Anfrageergebnis und als Schnittstelle für den Zugriff darauf.

Diese fünf Schritte beschreiben im Grunde die aufeinander folgenden Verarbeitungsphasen einer Anfrage. Der daraus folgende Ablauf ist in Abb. 4.12 nochmals graphisch dargestellt.

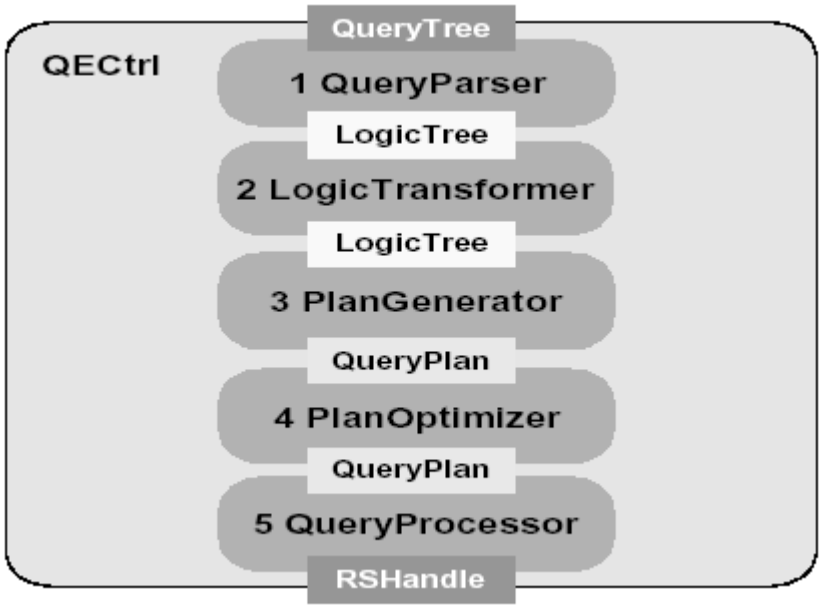


Abb. 4.12: Phasen der Anfrageverarbeitung

Die Kontrolle des Ablaufs obliegt dem Kontrollmodul (*QECtrl*). Diese Enterprise-JavaBean wird von den Komponenten des WEB-Tier benutzt und liefert ein Objekt für die Repräsenation des Ergebnisses zurück.

4.4.2 Interne Repräsentation und Verarbeitung einer Anfrage

Die AV erhält die Anfrage in dem schon zuvor vorgestellten speziellen Format. Dieser Suchbaum („QueryTree“) bietet dem Anwender eine relativ einfache, Schema-unabhängige Möglichkeit, eine Anfrage an die Datenbank zu stellen. Ferner bietet sie die Möglichkeit, die Volltextsuche unkompliziert zu integrieren. Die Einschränkung dieser besonderen Art der „Anfragesprache“ liegt darin, dass nicht mehr Möglichkeiten als die Formulierung einer Selektion gegeben sind. Die Struktur des QueryTree ermöglicht es aber, einfache Prädikate mit Hilfe der XML-Elementen zu bilden. Um auch komplexere Anfragen formulieren zu können, besteht die Möglichkeit mehrere solcher Prädikate mit den logischen Operatoren AND, OR und AND NOT zu Ausdrücken zu kombinieren.

Um die Anfrage effizient und einfach verarbeiten zu können, wird sie in eine entsprechende interne Repräsention umgewandelt. Die interne Darstellung („LogicTree“) einer Anfrage ist, wie die Anfrage selbst, als binärer Baum realisiert: mit Prädikaten als Blätter und Operatoren als Knoten. Die zur Verfügung stehenden logischen Operatoren AND, OR und NOT bilden eine Basis, d.h., jeder logische Term ist somit als ein LogicTree darstellbar. Diese Eigenschaft ist vor allem wichtig für spätere Transformationen der internen Darstellung.

Der QueryParser hat nun die Aufgabe, für einen gegebenen QueryTree einen äquivalenten LogicTree zu erzeugen. Die binäre Baumstruktur erlaubt es einen recht einfachen rekursiven Algorithmus zu formulieren, der diese Arbeit durchführt. Der Algorithmus erstellt für einen übergebenen Konten des QueryTree einen entsprechenden Knoten des LogicTree.

Die logische Transformation stellt verschiedene Methoden zur Verfügung, um die Interndarstellung in eine geeignete Normalform zu bringen. Im Wesentlichen sind dies die konjunktive (KNF) und die disjunktive (DNF) Normalform. Je nach Struktur der Anfrage kann eine solche Transformation die spätere Optimierung begünstigen. Die im LogicTransformer verwendeten Algorithmen sind alle in [Mit95] zu finden. Die Normalisierung basiert auf drei Regeln:

- Ersetze jedes Vorkommen von **Not(Not(a))** durch **a**.
- Ersetze jedes Vorkommen von **Not(And(a, b))** durch **Or(Not(a), Not(b))**.
- Ersetze jedes Vorkommen von **Not(Or(a, b))** durch **And(Not(a), Not(b))**.

Die Transformation in die konjunktive Normalform lässt sich also rekursiv formulieren:

- Ersetze jedes Vorkommen von **Or(a, And(b, c))** durch **And(Or(a, b),Or(a, c))**.
- Ersetze jedes Vorkommen von **Or(And(a, b), c)** durch **And(Or(a, c), Or(b, c))**.

Für die disjunktive Normalform gilt analog:

- Ersetze jedes Vorkommen von **And(a, Or(b, c))** durch **Or(And(a ,b), And(a, c))**.
- Ersetze jedes Vorkommen von **And(Or(a, b), c)** durch **Or(And(a, c), And(b, c))**.

Der Anfrageplan ist eine Repräsentation der späteren SQL-Anfrage, der allerdings eine einfache Manipulation und Analyse zulässt. Das Erzeugen des Anfrageplans ist einer der komplexesten Schritte der Anfrageverarbeitung. In dieser Phase findet die Zuordnung von XML-Elementen und Datenbank-Tabellen bzw. DB-Attributen sowie die Vorbereitung, Ausführung und Aufbereitung der Volltextsuche statt.

Die Metadaten-Attribute der erfassten Lehr-/Lern-Materialien (oder auch Ressourcen) werden in einem relationalen Schema verwaltet, wobei die einzelnen Merkmale in verschiedenen Tabellen gespeichert sein können und über Beziehungen den jeweiligen Ressourcen zugeordnet werden können. Die zu erzeugende SQL-Anfrage soll nun die Schlüssel der Datensätze liefern, deren Merkmale die vom *LogicTree* beschriebenen Eigenschaften erfüllen. Um nun eine entsprechende Anfrage generieren zu können, muss das Schema der Datenbank und die Abbildung der XML-Elemente auf die SQL-Tabellen bzw. Attribute in einer geeigneten Form verfügbar gemacht werden. Diese Informationen werden durch ein so genannte Metamodell beschrieben.

Es handelt es sich dabei auch um ein relationales Schema, welche die Zuordnung von XML-Attributen und Tabellenspalten sowie die Beziehung zwischen den einzelnen Attributen beschreibt. Die Struktur dieses Schemas ist aus dem Diagramm in Abb. 4.13 ersichtlich.

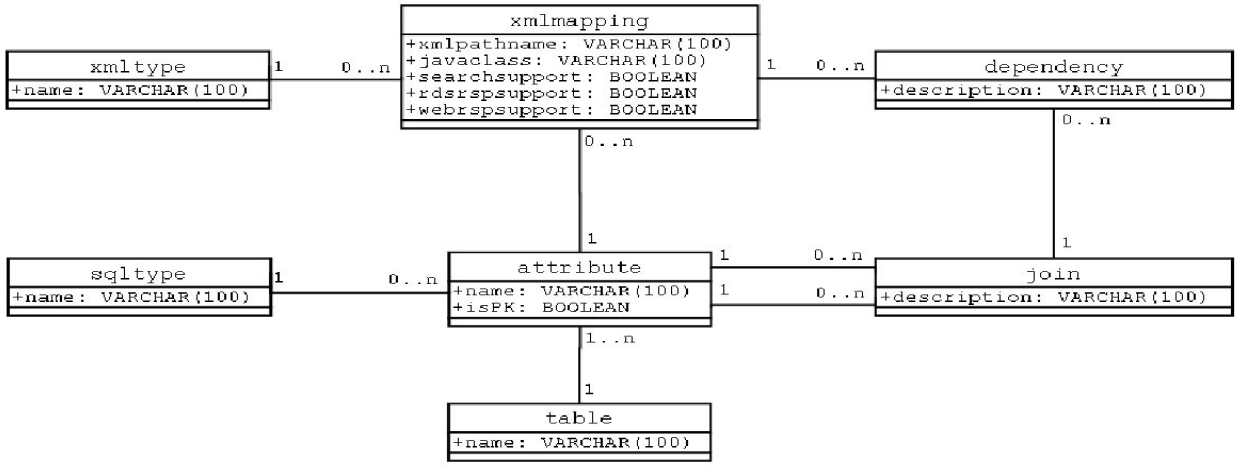


Abb. 4.13: Metamodell-Schema

Um die Volltextsuche möglichst generisch in das System zu integrieren, wurde beschlossen, das Ergebnis des Suche als temporäre Tabelle in der relationalen Datenbank zu materialisieren. Diese Vorgehensweise ermöglicht es, das Ergebnis der Volltextsuche direkt in die SQL-Anfrage einzubeziehen, ohne dass ein externes Zusammenführen der Ergebnisse erforderlich wäre. Ferner können Suchergebnisse gespeichert und für andere Anfragen wiederverwendet werden („Caching“).

Da die Merkmale der Ressourcen in einer relationalen Datenbank mit mehreren Tabellen gespeichert werden, ist es in vielen Fällen nötig, ein Kreuzprodukt („Join“) der beteiligten Tabellen zu erstellen und die interessanten Zeilen mit einer entsprechenden Verknüpfungsvorschrift („Join pre-

dicate“) auszuwählen. Die Information, welche Tabellen unter welchen Bedingungen an einem Join teilnehmen, liefert auch das Metamodell.

Bei der Formulierung der Anfrage ist jedoch zu beachten, dass eine einfache Verknüpfung aller beteiligten Tabellen in einem einzigen *Join* nur in Ausnahmefällen zum gewünschten Ergebnis führt. Eine solche Anfrage würde nur Datensätze liefern, für die in jeder beteiligten Tabelle mindestens ein Datensatz vorhanden ist, der die Join-Bedingung erfüllt. Da jedoch nicht für alle Dokumente jedes mögliche Merkmal erfasst wird, können Datensätze aus dem Ergebnis herausfallen, obwohl sie den vom Benutzer festgelegten Kriterien genügen. Auch die Benutzung von äußeren Verbundoperatoren ist nicht einfach möglich. Einen weiteren Stolperstein bilden die 1:N- oder M:N-Beziehungen. Dies wird allerdings bei der Generierung der SQL-Anfrage berücksichtigt. Zuvor ist der Anfrageplan jedoch noch zu optimieren.

Das Ziel der Anfrageoptimierung durch eine geeignete Komponente („PlanOptimizer“) besteht darin, den Anfrageplan in eine für die Ausführung möglichst günstige Form zu bringen. In diesem speziellen Fall bedeutet dies das Zusammenfassen von mehreren Unteranfragen zu möglichst nur einer Anfrage. Dabei müssen jedoch wiederum die Besonderheiten des Schemas beachtet werden, die bereits kurz erläutert wurden. Diese Eigenschaften führen zu zwei wichtigen Regeln für die Optimierung:

- Fasse alle Disjunktionen (OR) zusammen, welche die gleichen Tabellen betreffen.
- Fasse alle Konjunktionen (AND) zusammen, welche nicht die gleichen Attribute betreffen

Da der PlanOptimizer keine logischen Umformungen vornimmt, kann es von Vorteil sein, die Anfrage schon im Hinblick auf die Optimierung zu formulieren oder vorher entsprechende logische Transformationen durchzuführen. Negationen können allerdings nicht so einfach zusammengefasst werden, da dies in der Regel zu komplexe Umformulierungen der Anfrage erfordern würde.

Die letzte Phase der Anfrageverarbeitung ist die Ausführung der Anfrage. Dabei wird aus dem Anfrageplan eine SQL-Anfrage generiert, diese an die Datenbank gestellt und das Ergebnis in einer temporären Tabelle gespeichert. Diese Tabelle wird registriert und ein eindeutiger Schlüssel zur Verfügung gestellt, der externen Moduln des Systems (beispielsweise der Ergebnisverwaltung) den Zugriff auf das Resultat der Suche ermöglicht. Überdies werden den Datensätzen, sofern vorhanden, Volltextauszüge und eine fortlaufende Nummer, abhängig von der Volltextsuche, zugeordnet, um die Ergebnisse entsprechend gewichten zu können und gemäß des Stateless-Cursor-Konzeptes darauf zugreifen zu können, was eine fortlaufende Nummerierung der Ergebnisse voraussetzt.

4.5 Ähnlichkeits-basierte Suche

Die oben genannte Anfrageverarbeitung kann ferner durch eine Ähnlichkeits-basierte Suche weiter unterstützt werden. Diese Art von Suche findet man sehr häufig beispielsweise im Bereich des E-Commerce, des Fall-basierten Schließens, Wissensmanagements oder auch Text-Retrieval. Die In-

tegration der Ähnlichkeits-basierte Suchen in ein Datenbanksystem (DBS) kann immense Vorteile bringen, da kostenintensive Funktionsaufrufe während der DB-internen Anfrageverarbeitung vermieden werden können. Durch eine Integration können ferner die Kommunikationskosten drastisch gesenkt werden, da nicht mehr alle Daten zum Client transportiert werden müssen, sondern interne Berechnungen durchgeführt werden können. Weitere Vorteile verschafft die Möglichkeit, spezielle Indexstrukturen innerhalb des DBS nutzen zu können. Die Suche kann dadurch sehr effizient ausgeführt werden. Ähnlichkeits-basierte Suche lässt sich also sehr gut einsetzen, um die Rate der Dokumente, die relevant sein können, zu steigern, ohne stark erhöhte Antwortzeiten in Kauf nehmen zu müssen [Hau02, HMR02].

4.6 Semi-automatische Erschließung

Ein großes Problem allerdings ist nach wie vor die Erschließung großer Sammlungen von Dokumenten. Hier können geeignete Werkzeuge viele der anfallenden Arbeiten erleichtern oder sogar vollständig übernehmen. Solche Werkzeuge sollten sich dadurch auszeichnen, dass sie nicht nur komfortable Formulare und graphische Benutzeroberflächen anbieten, sondern insbesondere dadurch, dass sinnvolle Vorschläge für das Ausfüllen solcher Formulare gemacht werden. Es ist denkbar, dass Namen erkannt, aus den zu erschließenden Dokumenten extrahiert und dem Benutzer vorgeschlagen werden. Dabei möchte man durch die Verwendung von Text-Mining-Techniken zu solchen Vorschlägen gelangen.

4.6.1 Informationsextraktion

Bei der Informationsextraktion bzw. beim Text-Mining geht es um das Aufspüren und Strukturieren relevanter Informationseinheiten aus einer Menge von unstrukturierten oder semi-strukturierten Texten. Ein wichtiger Teilbereich hierbei beschäftigt sich mit der Erkennung so genannter „Named-Entities“. In den letzten Jahren werden in diesem Teilbereich verstärkt maschinelle Lernverfahren eingesetzt. Viele Text-Mining-Algorithmen wurden entwickelt, um Namen in unstrukturierten Dokumenten zu finden und zu klassifizieren. Unter „Named-Entities“ [Ste01] werden auch Personennamen verstanden. Personennamen in einem Dokument zu erkennen ist eine schwierige Aufgabe. Normalerweise kann man nur, wenn man die interne Struktur des Satzes oder den Kontext genau betrachtet, entscheiden, ob ein Satzteil ein Personenname ist. Allein durch Betrachtung einzelner Wörter kann dies nicht entschieden werden.

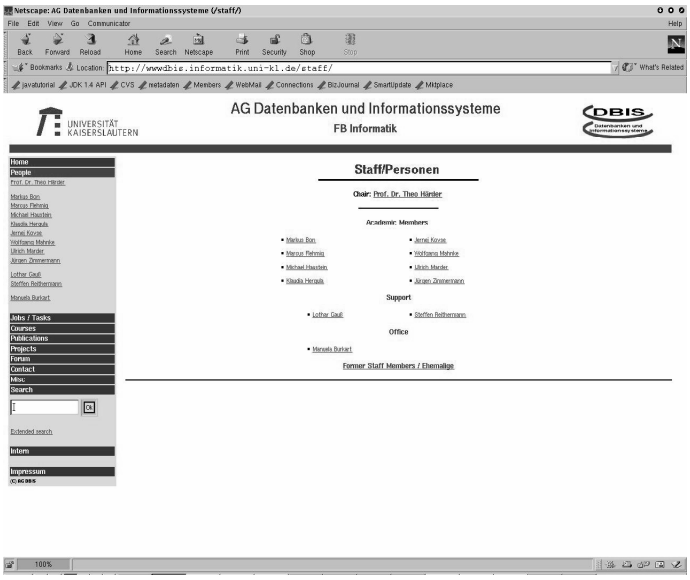


Abb. 4.14: Beispielseite

Sehr hilfreich ist hierbei der Einsatz von Wörterbüchern bzw. Datenbanken. Hierbei stößt man allerdings auf zwei Probleme: Zum einen dass die Wörterbücher nie vollständig sein können, da die Anzahl der Personennamen nicht beschränkt ist und diese Namen in verschiedenen Formen vorkommen; zum anderen dass die Regeln und Heuristiken, die verwendet werden müssen, nicht alle Fälle abdecken können. Ferner führt der Umstand, dass im Deutschen keine Unterscheidung zwischen Substantiven und Namen in Bezug auf ihre Schreibweise gemacht wird, zu weiteren Problemen.

4.6.2 Text-Mining-Konzepte

Ähnlich wie Data Mining die Analyse strukturierter numerischer Daten kennzeichnet, beschreibt der Begriff des Text Mining eine Menge von Methoden zur (halb-)automatischen Auswertung großer Mengen natürlichsprachlicher Texte. Das Gebiet des Text Mining umfasst vielfältige Methoden zur Extraktion von Informationen aus natürlichsprachlichen Texten.

In diesem Gebiet bzw. in dem Forschungsgebiet der Namenserkennung und Namensklassifikation werden viele Algorithmen vorgestellt und dokumentiert. Bei diesen Namenerkennungssystemen, die Teile der so genannten „Information Extraction Systems“ sind, werden hauptsächlich zwei Ansätze verwendet und zwar der sogenannte „Knowledge Engineering Approach“ und „Automatic Training Approach“ [AI99]. Der hier verfolgte Ansatz folgt dem „Knowledge Engineering Approach“, indem Wörterbücher und Heuristiken benutzt werden, um Namen in Dokumenten zu erkennen.

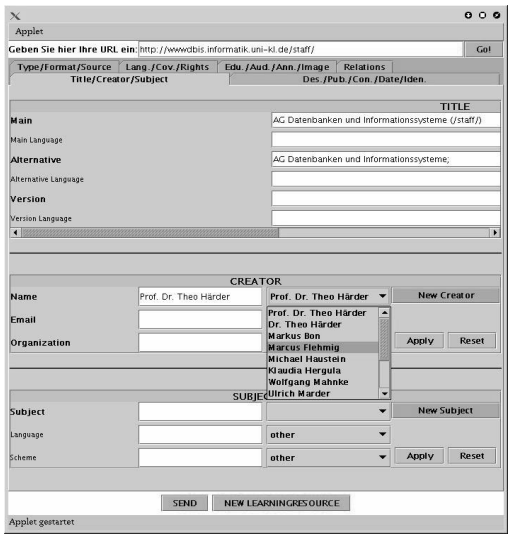


Abb. 4.15: Beispielanzeige des semi-automatischen Erschließungswerkzeuges

4.6.3 Beispiel

Beispiel aus einem HTML-Dokument mit Personennamen, die teilweise mit akademischen Titeln eingeführt werden (Abb. 4.14). In diesem Fall werden mit Hilfe von Vor- und Nachnamenwörterbücher verschiedene Heuristiken angewendet. Das Ergebnis als Vorschlag in den Feldern des semiautomatischen Erschließungswerkzeuges sieht man in Abb. 4.15.

4.7 Automatische Erschließung und Verbesserung der Qualität der Erschließung bei schlecht erschlossenem Material

Nachdem erste Versuche bereits gezeigt hatten, dass ein statistischer Ansatz mit zu Hilfenahme von Wortlisten zur automatischen Klassifikation nach der Regensburger Verbund Klassifikation (RVK)[10] nicht die gewünschten Ergebnisse erbringt, wurde über ein alternatives Verfahren nachgedacht [Ber02]. Dabei wurde auch nach einer Möglichkeit gesucht, die automatische Beschlagwortung mit Vokabular der SWD (Schlagwortnormdatei) zu integrieren. Bei den Überlegungen, welcher neue Ansatz zur Klassifizierung und Beschlagwortung gewählt wird, wurde darauf geachtet, dass man nach Möglichkeit ohne Wörterbücher auskommt, da es sehr aufwändig ist, diese zu erstellen. Außerdem sollte ein neuer Ansatz ohne große Anpassung auf verschiedene Sprachen anwendbar sein. Aus diesen Gründen wurde ein Lernverfahren ausgewählt, um aus bereits klassifizierten und beschlagworteten Dokumenten die Informationen, die zur Klassifizierung und Beschlagwortung neuer Dokumente benötigt werden, zu extrahieren. Als eines der besten Lernverfahren hat sich die sogenannte „Support Vector Machine“ (SVM) herausgestellt. Das Lernverfahren „Support Vector Machine“ ist ein noch recht junges Verfahren, das bereits in vielen Anwendungsgebieten die meisten anderen Systeme übertroffen hat. SVM-Verfahren werden nicht nur zur Klassifizierung von Texten genutzt, sondern finden auch in Bereichen der Klassifizierung von Bildern, Schrifterkennung, Objekterkennung und vielen anderen Bereichen Verwendung [Web02].

4.7.1 Beschreibung des Klassifizierungs- und Beschlagwortungssystems

Abb. 4.16 zeigt den Arbeitsablauf des neuen Klassifizierungs- und Beschlagwortungssystems nach:

1. **Import neuer Dokumente**

   Im ersten Arbeitsschritt wird eine Anfrage an den Controller der zentralen Datenverwaltungskomponente (DVK) gestellt, um alle neuen, noch nicht klassifizierten oder beschlagworteten Dokumente zu suchen. Dieser liefert dann alle relevanten Daten, die für die Klassifizierung und Beschlagwortung dieser Dokumente notwendig sind.

2. **Import der Lernmenge**

   Hier werden die speziell ausgewählten Dokumente, die typisch für bestimme Klassen und Schlagwörter sind, importiert.

10.) http://www.bibliothek.uni-regensburg.de/Systematik/systemat.html

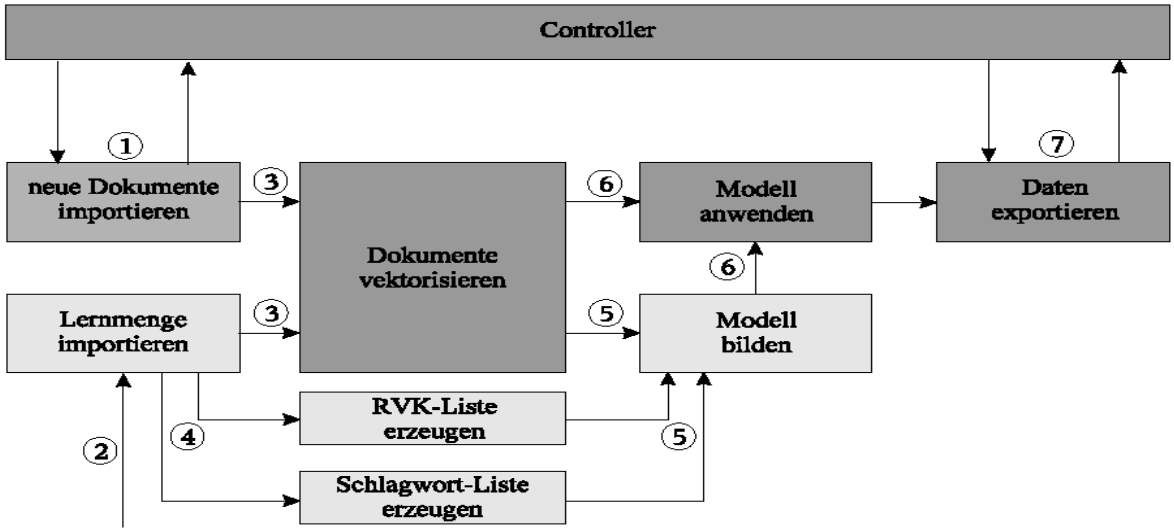


Abb. 4.16: Arbeitsablauf

3. Dokumente vektorisieren

Um mit den bereits importierten Dokumenten arbeiten zu können, müssen diese erstmal in die Vektordarstellung überführt werden. In Abb. 4.17 ist die Konvertierung eines Dokumentes in eine Vektorrepräsentation dargestellt. Danach müssen die Wörter nach der Häufigkeit des Vorkommens in den einzelnen Dokumenten gewichtet werden.

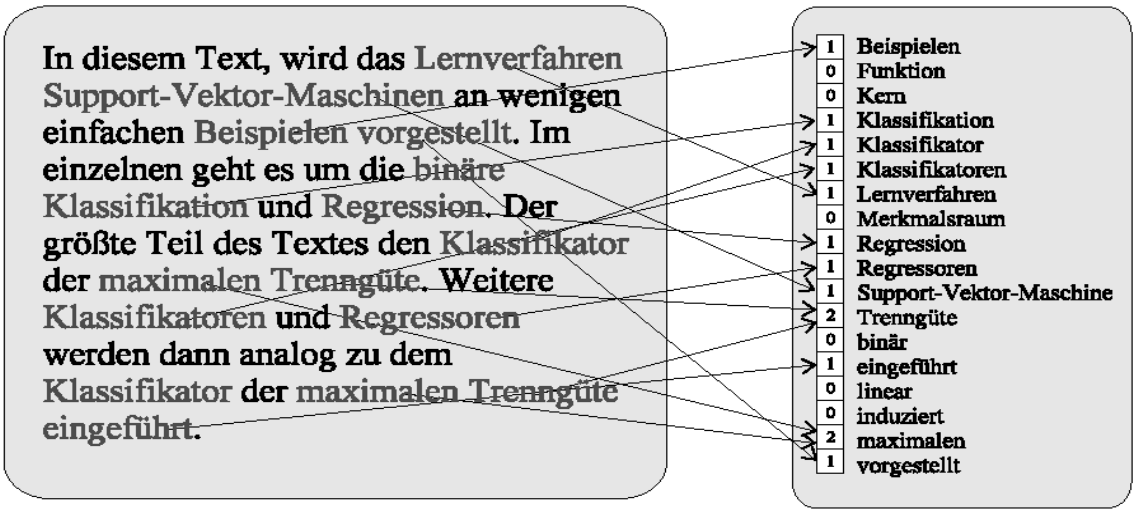


Abb. 4.17: Attribute-Wert-Darstellung eines Dokumentes

4. Listen erzeugen

In diesem Arbeitsschritt wird aus der Lernmenge, die im zweiten Schritt importiert wurde, eine Liste aller in der Lernmenge vorhandenen Klassifikationen und Schlagwörter bestimmt. Dieses Verfahren ist leider nur in der Lage, Klassifikationen und Schlagwörter zu vergeben, die in der Lernmenge enthalten sind.

5. Modell bilden

Für jede Klassifikation und jedes Schlagwort muss ein Modell gebildet werden, mit dem man für neue Dokumente entscheiden kann, ob sie zu einer bestimmten Klasse gehören oder nicht. Dabei teilt man für jede Klasse die Lernmenge in eine Positiv- und eine Negativmenge auf. Ziel ist es nun eine Funktion zu finden, die diese beiden Mengen linear trennt. SVM bildet ein System zum Lernen von linearen Funktionen in einem kerninduzierten Merkmalsraum unter Berücksichtigung der Ergebnisse der Generalisierungstheorie und der Ausnutzung der Optimierungstheorie. Man versucht lineare Funktionen zu nutzen, da sie recht gut erforscht und einfach realisierbar sind. Jedoch sind lineare Funktionen nicht geeignet, um reale Probleme zu lösen. Daher geht man in einen hochdimensionalen kerninduzierten Merkmalsraum über, in dem viele Probleme linear trennbar werden.

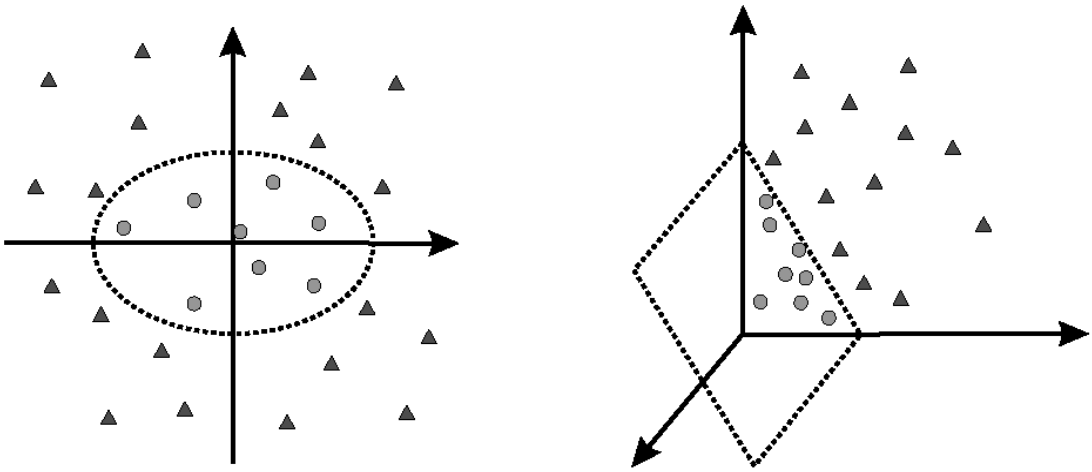


Abb. 4.18: Transformation in einen höherdimensionalen Merkmalsraum

In Abb. 4.18 ist die Überführung in einen höherdimensionalen Merkmalsraum und die Trennung durch eine lineare Funktion dargestellt. In der Anwendung entstehen oft Vektoren mit mehr als 10000 Einträgen.

6. Modell anwenden

Für alle neuen Dokumente muss nun in jedem Modell getestet werden, ob sie zur Positiv- oder zur Negativmenge gehören. Falls ein Dokument zur Positivmenge gehört, wird die entsprechende Eigenschaft des Models gespeichert.

7. Daten exportieren

Nun wird jedes Dokument, für das eine Klassifikation oder ein Schlagwort gefunden wurde, von der DVK im XML-Format angefordert. Dann wird das XML-Dokument um die neuen Eigenschaften ergänzt und wieder an die DVK zurückgeschickt.

4.7.2 Evaluierung

Die bisherigen Testläufe wurden mit deutschsprachigen Dokumenten aus dem Fachgebiet Mathematik durchgeführt. Dabei wurde aus der vorläufigen Linksammlung eine Lernmenge von ca. 700 Dokumenten entnommen. Als Testmenge wurden ca. 170 neue Dokumente gesucht und intellektuell erschlossen. Die im Arbeitsschritt 5 beschriebene Modellbildung kann auf unterschiedliche Weise erfolgen. Bisher wurden zwei unterschiedliche Ansätze untersucht:

1. Virtuelle Klasse

Bei diesem Ansatz wurden jeweils Kombinationen von Klassifikationen zu einer neuen virtuellen Klasse vereint. Die SVM wurde dann dazu benutzt, um eine Klasse aus der Menge der virtuellen Klassen auszuwählen. Die ausgewählte virtuelle Klasse wurde dann wieder in ihre einzelnen Klassifikationen zerlegt. Die Beschlagwortung erfolgte analog.

2. Binäre Entscheidung

In dem Fall „Binäre Entscheidung“ wird für jede Klassifikation und für jedes Schlagwort einzeln an Hand einer Positiv- und Negativmenge untersucht, ob eine Klassifikation oder ein Schlagwort vergeben werden soll.

	Klassifikation: precision	Klassifikation: recall	Beschlagwortung: precision	Beschlagwortung: recall
Virtuelle Klasse	80.3%	59.0%	80.1%	62.4%
Binäre Entscheidung	89.2%	52.7%	90.4%	58.4%

Tab. 4.19: Vergleich von Ansatz 1 und 2

Der Versuch, die Lernmenge mit Daten des Bibliotheksverbundes Bayern (BVB) auf ca. 4000 Dokumente zu vergrößern, brachte eine Verschlechterung des Ergebnisses bei der Klassifikation. Die Precision sank auf 65.2% und der Recall auf 35.9%. Auf Grund dieser Ergebnisse vermuten wir, dass die Struktur der Titeldaten bei Online-verfügbarem Lern- und Lehrmaterial nicht mit dem Datenbestand des BVB vergleichbar ist, denn die Anwendung der Lernmenge auf BVB-Daten brachte wieder bessere Ergebnisse [Web02].

5. Diplom- und Projektarbeiten

Im Jahr 2002 wurden 8 Projektarbeiten betreut, davon wurden 4 im laufenden Jahr abgeschlossen und weitere 3 werden noch zu Beginn des neuen Jahres beendet sein. Ferner wurden 5 Diplomarbeitsthemen ausgegeben. Zwei der betreuten Diplomarbeiten wurden bereits erfolgreich beendet.

6. Literatur

Eigene Literatur

Aml02 Amlinger, C.:
Komponente zur Web-Service-unterstützten Integration von XML-Dokumenten in ein relationales DBS im Rahmen des Projekts Meta-Akad, Projektarbeit, FB Informatik, Universität Kaiserslautern, noch nicht abgeschlossen.

Ber02 Berscheid, F.:
Ablaufkoordination, automatisiertes Sammeln und Erkennen von Lehr-/Lernmaterialien im Projekt META-AKAD, Diplomarbeit, FB Informatik, Universität Kaiserslautern, Mai 2002.

Boh02 Bohler, T.:
Spezifikation und Realisierung einer XML-Partition-Miner-Komponente für SHARX, Diplomarbeit, FB Informatik, Universität Kaiserslautern, noch nicht abgeschlossen.

Büh02a Bühmann, A.:
Möglichkeiten der Extraktion von Schemainformationen aus XML-Dokumenten und deren formale Beschreibung zur Verwendung in SHARX, Projektarbeit, FB Informatik, Universität Kaiserslautern, Juli 2002.

Büh02b Bühmann, A.:
Konzepte für die Einbettung bzw. Integration von XML und XQuery in eine Wirtssprache (Java) Diplomarbeit, FB Informatik, Universität Kaiserslautern, noch nicht abgeschlossen.

Cha02 Chatti, A.:
Einsatz von Text-Mining-Algorithmen bei der Realisierung eines semi-automatischen Erschließungswerkzeugs im Projekt Meta-Akad, Projektarbeit, FB Informatik, Universität Kaiserslautern, Dezember 2002.

Fle01a Flehmig, M.:
Data-Driven, XML-Based Web Management in Highly Personalized Environments, in: Proc. Int. Workshop on Information Integration on the Web (WIIW'2001), Rio de Janeiro, Brazil, April 2001, pp. 81-88.

Fle01b Flehmig, M.:
Integration und Personalisierung - Zur Realisierung essentieller Aspekte in Portal-Systemen, in: Tagungsband der GI/OCG-Jahrestagung Informatik 2001, Wien, Sept. 2001, pp. 895-901.

Fle02 Flehmig, M., Knüttel, H., Leiwesmeyer, B., Ritter, N., Schmettow, M., Weber, G.:
Virtuelle Lehre im Angebot der Universitätsbibliothek, in: Tagungsband der 16. DFN-Arbeitstagung über Kommunikationsnetze (Lecture Notes in Informatics - Proceedings P-17), Düsseldorf, Mai 2002, pp. 249-259.

FFW02 Flehmig, M., Fudeus, S., Weber, C.:
Realisierung einer Web-basierten Suchschnittstelle für die Verwendung in Meta-Akad, interner Praktikumsbericht, FB Informatik, Universität Kaiserslautern, August 2002.

Gau02 Gauß, B.:
Skalierungskonzepte in datenintensiven, mehrschichtigen und verteilten Anwendungensarchitekturen (J2EE) und ihre quantitative Bewertung (Projekt Meta-Akad), Diplomarbeit, FB Informatik, Universität Kaiserslautern, noch nicht abgeschlossen.

Gor02 Gorius, Ch.:
Analyse der Einsatzfähigkeit von XML zum Aufbau einer relationalen Datenbank, Diplomarbeit, FB Informatik, Universität Kaiserslautern, Mai 2002.

Hau02 Haustein, M.:
Ähnlichkeitssuche in objekt-relationalen Datenbanksystemen, Diplomarbeit, FB Informatik, Universität Kaiserslautern, Juni 2002.

HMR02 Haustein, M., Mahnke, W., Ritter, N.:
Index Techniques for Similarity-based Search in ORDBMSs, submitted for publication.

Höh02 Höh, D.:
SHARX - Ein XML-basiertes Web-Site-Management-System: Modul Repository, Projektarbeit, FB Informatik, Universität Kaiserslautern, Juni 2002.

Sch02 Schultz, A.:
Spezifikation einer Registrierungskomponente für XML-Artefakte im Bereich personalisierter Webumgebungen, Projektarbeit, FB Informatik, Universität Kaiserslautern, Juli 2002.

Sie01 Siegel, S.:
Flexible Kopplung heterogener autonom verwalteter Datenquellen an ein XML-basiertes Web-Site-Management-System - Konzepte und Implementierung, Diplomarbeit, FB Informatik, Universität Kaiserslautern, Dezember 2001.

Wag02 Wagner, B.:
XML-basierte Anfrageverarbeitung der Datenverwaltungskomponente im Rahmen des Projekts Meta-Akad, Projektarbeit, FB Informatik, Universität Kaiserslautern, noch nicht abgeschlossen (vorauss. Jan. 2003).

Web02 Weber, C.:
Realisierung einer automatischen Klassifizierungs- und Beschlagwortungskomponete im Rahmen des Projekts META-AKAD, Projektarbeit, FB Informatik, Universität Kaiserslautern, noch nicht abgeschlossen (vorauss. Jan. 2003).

Zen02 Zendel, O.:
Leistungsbewertung von XML Publishing Frameworks, Projektarbeit, FB Informatik, Universität Kaiserslautern, noch nicht abgeschlossen (vorauss. Jan. 2003).

Weitere Literatur

ABS00 Abiteboul, S., Buneman, P., Suciu, D.:
Data on the Web - From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers, San Francisco, California, 2000.

AI99 Appelt, D. E., Israel, D. J.:
Introduction to Information Extraction Technology, Tutorial for IJCAI-99, Stockholm, Sweden, 1999

BS95 Borghoff, U. M., Schlichter, J. H.:
Rechnergestützte Gruppenarbeit - Eine Einführung in Verteilte Anwendungen, Springer- Lehrbuch, Berlin Heidelberg, Deutschland, 1995.

CFP99 Ceri, S., Fraternali, P., Paraboschi, S.:
Data-Driven, One-to-One Web-Site Generation for Data-Intensive Applications, in: Proc.of 25th Int. Conf. on Very Large Data Bases, Edinburgh, Scotland, UK, 1999, pp. 615-626.

Gri98 Griffel, F.:
Componentware- Konzepte und Techniken eines Softwareparadigmas, dpunkt-Verlag, Heidelberg, Deutschland, 1998

GW97 Goldman, R. , Widom, J.:
DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases, in: Proc. of 23rd International Conference on Very Large Data Bases (VLDB'97), Athens, Greece August 1997, pp. 436 445.

Mit95 Mitschang, B.:
Anfrageverarbeitung in Datenbanksystemen — Entwurfs- und Implementierungskonzepte, Reihe Datenbanksysteme, Vieweg, 1995.

Ste01 Steiner, I.:
Warum "Named Entities" für die Chunk-Analyse wichtig sind, in: Proc. der GLDV-Frühjahrstagung 2001, Henning Lobin (Hrsg.), Universität Gießen, März 2001, pp. 245-252.

Suc98 Suciu, D.:
Semistructured Data and XML, in: 5th International Conference of Foundations of Data Organization (FODO'98), Kobe, Japan, 1998.