Erweiterung und Nutzung neuer Informationstechniken beim Web-basierten Lehren und Lernen

Abschlussbericht

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

August 2004

1 Einführung

Um mit der künftigen Entwicklung auf dem Gebiet des netzbasierten Lehrens und Lernens Schritt zu halten, bedarf es vermehrter Anstrengungen zur Nutzung neuen Informations- und Kommunikationstechnologien sowie ihre Integration in entsprechende Lehr-/Lernumgebungen. 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. Der Übergang von papierbasierten hin zu digitalen, multimedialen Informationsangeboten setzt die Schaffung einer geeigneten Infrastruktur voraus, die sowohl anspruchsvollen technischen als auch wichtige sicherheitsrelevanten Aspekten genügen muss. Erste Erfahrungen zeigen, dass derartige Informationssysteme in der Regel umfangreiche dynamische Inhalte besitzen, für die ein hoher Grad an Automatisierung bei der Verwaltung, Wartung und Aktualisierung anzustreben ist. Aus diesen Gründen sollte die Administration und Bereitstellung der Informationsangebote eines solchen Systems datenbankbasiert erfolgen. Deshalb versucht unser Forschungsprojekt „Erweiterung und Nutzung neuer Informationstechniken beim Web-basierten Lehren und Lernen“ Kompetenz in den Bereichen „Web-basierte DB-Anwendungen“ einzusetzen, um substanzielle Verbesserungen bei der Abwicklung und Kontrolle der zu unterstützenden Funktionen zu erzielen. Hierbei geht es neben der Realisierung eines Web-Informationssystems vor allem um die Individualisierung von Lernumgebungen und die Verwaltung der benötigten Lehrdokumente, wobei eine Reihe von technischen Problemen (Kommunikation, Sicherheit) zu lösen sind. Durch prototypische Realisierung von (Teil-) Systemen wird zugleich eine breite Know-how-Bildung in diesem sich ernorm schnell entwickelnden Bereich der angewandten Informatik angestrebt. Dieser Bericht beschreibt die Ergebnisse der Arbeiten des Projektes, das seit August 2000 von der Stiftung Rheinland-Pfalz für Innovation gefördert wird. Die vorbereitenden Arbeiten begannen in der Arbeitsgruppe Datenbanken und Informationssysteme bereits im September 1999.

1.1 Gliederung

In Kapitel 2 werden unsere Arbeiten zu Infrastruktur- und Sicherheitsaspekten bei konkreten Web-basierten Anwendungen skizziert. Es sind dabei eine Reihe von Web-basierten Systemen entstanden, die ihren Härtetest im Lehrbetrieb des Fachbereichs bereits bestanden haben. Breiten Raum nimmt das Forschungsthema Individualisierung/Personalisierung von Dokumenten ein, dessen Ergebnisse in Kapitel 3 abgehandelt werden. Erst in den letzten drei bis vier Jahren hat sich in diesem Zusammenhang der Begriff der Portalarchitekturen herausgebildet. Zu ihrer Konkretisierung und der Bestimmung ihres Anforderungsspektrums wurden eine Reihe von Arbeiten durchgeführt, deren Ergebnis eine mehrschichtige Systemarchitektur ist, die einen „Unified View” auf die gesamte Dokumentenbasis bietet und ein umfassendes Personalisierungskonzept zu realisieren gestattet. Viele Entwurfsentscheidungen und -verfeinerungen, die nachfolgend beleuchtet werden, führten auf eine Komponentenarchitektur, deren Realisierung, vor allem durch studentische Arbeiten, bereits weit fortgeschritten ist. Unsere Arbeiten zum Thema Dokumentenverwaltung sind in Kapitel 4 und 5 zusammengefasst. Wiederverwendung von Elementen einer virtuellen Lernumgebung ist auf

verschiedenen Ebenen denkbar. Es können Lerninhalte, strukturierte Aggregationen von Lerninhalten oder ganze Kurse, inklusive ihrer Präsentation, wiederverwendet werden. Als grundlegendes Konzept wurde von uns zunächst untersucht, wie sich Dokumente aus Fragmenten aufbauen und redundanzfrei verwalten lassen. Dazu haben wir ein Repositorium XCoP (XML Content Repository) auf der Basis eines objekt-relationalen Datenbanksystems entwickelt. XCoP wartet die Strukturinformation von Dokumenteninhalten und verwaltet die Beziehungen zwischen den einzelnen Dokumentfragmenten. Es verbessert durch Nutzung dieser Strukturinformation die inhaltliche Verarbeitung von XML-Dokumenten. So ist eine Wiederbenutzung von Dokumentfragmenten möglich, um Datenredundanz und als Folge davon Änderungsanomalien bei replizierten Daten zu vermeiden. Die Erkenntnisse des XCoP-Repositoriums gingen bei dem Aufbau eines Web-basierten Nachweissystems für Online-verfügbare Lehr- und Lernmaterialen ein. Dieser Dienst befindet sich im Produktivbetrieb. Als weiteren Aspekt der Dokumentenverwaltung haben wir die inhaltsbezogene Suche in Textdokumenten, insbesondere die Ähnlichkeitssuche, in Angriff genommen. Dazu sollen DB-basierte Maßnahmen zu ihrer Unterstützung und Verbesserung entwickelt und evaluiert werden. Die Grobarchitektur für eine Systemlösung sowie das Anforderungsspektrum für die zugehörige Datenverwaltung sind bereits fertiggestellt.

1.2 Veröffentlichungen, Diplom- und Projektarbeiten

Erfreulicherweise konnte die Arbeitsgruppe Datenbanken und Informationssysteme in der Projektlaufzeit nicht nur praktisch nutzbare Systeme und neue Konzepte zu aktuellen Forschungsfragen entwickeln, sondern auch in einer beachtlichen Reihe von Arbeiten ihre Ergebnisse publizieren. Außerdem konnten zahlreiche Arbeiten betreut und erfolgreich abgeschlossen werden. In den Jahren von 2000 bis 2003 konnten 22 Diplom- und Projektarbeiten betreut werden. Ferner wurden 2 projektspezifische Praktika durchgeführt.

2 Web-Technologien: Realisierte Anwendungen

Durch die explosionsartige Ausweitung des World-Wide Web (WWW) stellen sich täglich neue Herausforderungen an das Web-Management. War es anfangs noch leicht möglich, statische Dokumente auf einem zentralen Serversystem durch wenige Personen zu verwalten, so haben sich die Anforderungen an ein modernes Web-Management vollständig gewandelt. Es sind eine Vielzahl von verschiedenen Benutzern bis hin zu ganzen Unternehmen zu unterstützen, die Inhalte bereitstellen. Dabei stellt sich auch die Aufgabe, Datenbanken und andere Dokumenten-, Informations- bzw. Datenquellen, die einen starken heterogenen Charakter aufweisen können, zu integrieren und ihre Inhalte in einheitlicher Form zu präsentieren.

2.1 Anforderungen datenintensiver Web-basierter Anwendungen

Insbesondere bei datenintensiven Web-basierten Anwendungen können nicht alle Informationen in Form von statischen Dokumenten verwaltet oder vor einem ersten Zugriff vollständig generiert werden. Daten werden vielmehr derart in Web-Dokumente integriert, dass erst zum Zeitpunkt des Zugriffs das Dokument oder Teile des Dokuments, die aktuelle Informationen beinhalten, dynamisch generiert werden. Physisch existiert das Dokument unter Umständen nur zum Zeitpunkt der Anfrage. Da sich dadurch die in den Dokumenten enthaltenen Daten immer auf dem aktuellen Stand befinden, eignet sich dieser Ansatz besonders für änderungsintensive Anwendungsbereiche. Suchanfragen können dabei direkt auf der Datenbank ausgeführt werden.

Das spezifische Anforderungsspektrum von datenintensive Web-Anwendungen lässt sich deshalb wie folgt charakterisieren [CFP99]:

- Ausführung einfacher Funktionen
- Einfache Transaktionsstruktur der Programme
- Intensiver navigierender Zugriff mit adaptiver Benutzerschnittstelle
- One-to-One Delivery (Personalization)
- Unterstützung verschiedener endgeräteabhängiger Repräsentationsformen (Customization).

2.2 Paradigmenwechsel

Deklaratives Web-Site-Management ist als eines der neuen Paradigmen zur Realisierung solcher Anforderungen entstanden [FLM98]. Das Hauptmerkmal dieses Ansatzes besteht darin, dass die drei wichtigsten Aufgaben während der Entwicklung einer Web-Site identifiziert und entsprechend voneinander getrennt wurden: Datenverwaltung, Spezifikation der internen Struktur und Definition einer oder mehrerer Abbildungen auf entsprechende externe grafische Präsentationstrukturen [DAB+99]. Dabei erfolgt eine Abstraktion von der Menge der zugrunde liegenden, evtl. heterogenen Datenquellen auf ein logisches Modell, das Inhalt und Struktur einer Web-Site von der grafischen Repräsentation trennt.

Unterschiede in den verschiedenen Ansätzen finden sich häufig in dem zugrunde gelegten Datenmodell und in den Anfragesprachen, die benutzt werden, um Sichten für die Spezifikation der

Struktur und Präsentation auf diesem Datenmodell zu definieren. Erst die explizite Modellierung solcher Abstraktionsstufen ermöglicht eine umfassende Personalisierung (Personalization) von Daten, die im besonderen Maße im Bereich des E-Commerce [CFP99] und des „Web-Based-Teaching“ [DM98] wichtig ist. Neben der Anpassung von Inhalten und Struktur an die Bedürfnisse eines Anwenders (Individualization) eröffnet die Personalisierung auch die Möglichkeit der dynamischen Anpassung der Repräsentation (Customization). Gleiche Inhalte einer Datenbasis können dem Benutzer beispielsweise in Abhängigkeit des benutzten Clients (Web-Browser, WAP-Handy) unterschiedlich repräsentiert werden.

Anwendung im Bereich Web-Site-Management

Aus dem Paradigma des deklarativen Web-Site-Managements lässt sich der essenzielle Grundsatz der Trennung von Inhalt und Präsentation direkt ableiten. In vielen Anwendungen wird durch den alleinigen Einsatz der Seitenbeschreibungssprache HTML [HTML02] dieser Grundsatz schon verletzt. Die Vermischung der eigentlichen Daten und ihrer grafischen Präsentationsinformation macht eine Wartung nur sehr schwer möglich. Der Einsatz der *eXtensible Markup Language* (XML) [XML00] erlaubt es, die Repräsentation der Daten von den grafischen Präsentationsaspekten durch Kombination mit der *Extensible Stylesheet Language* (XSL) [XSL01] bzw. *XSL Transformations* (XSLT) [XSLT99] zu trennen. Hierdurch werden Redundanzen vermieden, da Daten aus einem XML-Dokument mehrfach verwendet werden können. Publikationslisten beispielsweise können nach Jahr oder nach Autor gruppiert oder sortiert werden. In der Beschreibung der verschiedenen externen grafischen Präsentationen ist keine Änderung notwendig, wenn die zugrunde liegenden Daten sich ändern oder erweitert werden. Änderungen an den Daten müssen auch nur an genau einer Stelle durchgeführt werden.

Bei der Neugestaltung von Teilen der Web-Seiten der AG DBIS fanden diese bereits in einigen Bereichen (Informationsdienst, Publikationen, Lehre) Berücksichtigung. Durch die Verwendung von XML konnten die Daten weitestgehend von der Struktur und Präsentation getrennt werden. Die Realisierung des Informationsdienstes als Mehrkanaldienst bietet beispielsweise die Möglichkeit der Integration sowohl von Nachrichten externer Dienste, so genannte Content Provider, als auch von verfallsbasierten Ankündigungen von Vorträgen, Seminaren oder Vorlesungsinformationen auf der Willkommensseite unserer Web-Site. Die Transformation der XML-Dokumente in ein entsprechendes Ausgabeformat geschieht durch die Unterstützung des *XML-Publishing Framework Cocoon* der Apache-XML-Gruppe [Maz00]. Im Rahmen einer Projektarbeit wurden weitere Produkte auf Einsatzfähigkeit und Leistungsfähigkeit hin untersucht [Zen02].

2.3 Web-basierte Anwendungen

Im Bereich Web-basierter Anwendungen setzt sich die Trennung von Daten, Struktur (bzw. Logik) und Präsentation in der Benutzung des Entwurfsmusters Model-View-Controller (MVC) fort (siehe Abb. 2.1). Viele Abläufe in einem virtuellen Lernbetrieb lassen sich in geeigneter Weise automatisieren. Besonders die aufwendigen administrativen Aufgaben der Verwaltung der Beteiligten in ihren unterschiedlichen Rollen, vom Interessenten über den Teilnehmer bis zum Absolventen, sollten

genauso unterstützt werden wie die Zahlungsabwicklung und die Implikationen im Hinblick auf die Nutzung von Lernangeboten. Es ergibt sich die Notwendigkeit eines umfassenden Autorisierungskonzeptes innerhalb eines solchen Systems, so dass beispielsweise eine Teilnahme an einem virtuellen Seminar erst nach Zahlung der Seminargebühren möglich wird. Aspekte des E-Commerce bzw. des E-Service sind hiermit verbunden.

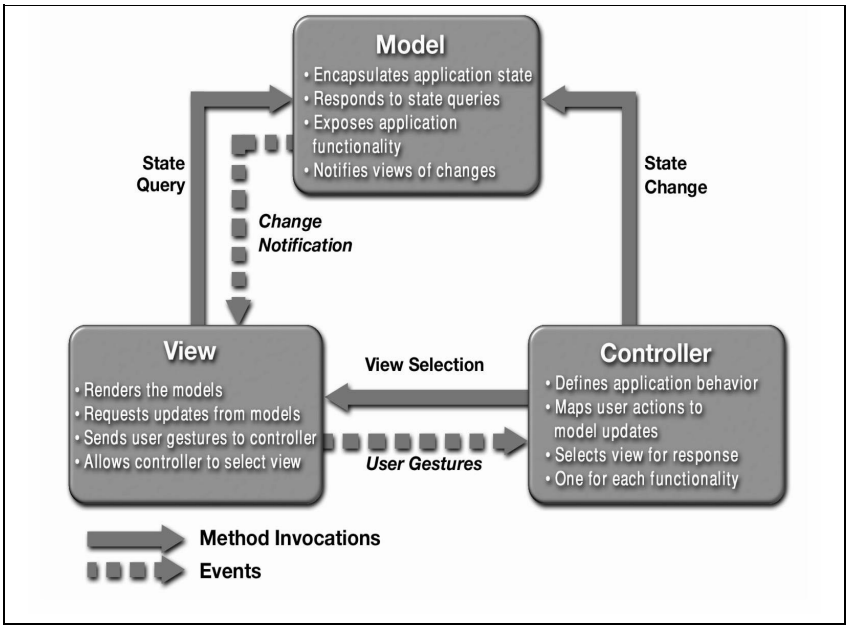


Abb. 2.1: Entwurfsmuster Model-View-Controller (MVC)

2.3.1 Prüfungsinformationssysteme: PAS und PAVIAN

Teile der oben genannten Funktionalität sind bereits durch unsere Systeme PAS (Prüfungs-Anmelde-System) und PAVIAN (Prüfungs-Administration VIA Netz) realisiert. Sie 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. Da bei all diesen Vorgängen sensitive, personenbezogene Daten involviert sind, musste großer Wert auf die Funktionen der Autorisierung, Authentifikation und Zugriffskontrolle gelegt werden. In Ermangelung von sicheren und praktisch anwendbaren Verfahren auf der Basis von biometrischen Daten musste ein sorgfältig entworfenes Passwortverfahren herangezogen werden. Das System wurde mit Hilfe des *Java 2 Enterprise Edition (J2EE) Application Framework* realisiert. Dabei wurde strikt auf die Einhaltung des MVC-Entwurfsmusters geachtet. Vorteile dieses Vorgehens lagen unter anderem darin, dass Änderungen der Präsentation keine Änderungen des Quelltextes oder gar eine erneute Übersetzung erfordern. Die verschiedenen Prüfungslisten werden bereits innerhalb des Fachbereichs Informatik eingesetzt und gestatten die Automatisierung vieler Routineaufgaben. Sie befinden sich 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.

2.3.2 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 hat das Ziel, Lernenden und Lehrenden einen einheitlichen Zugang zu einem umfassenden Angebot von im Internet verfügbaren Lehr- und Lernmaterial zu verschaffen. Der Zugang wird seit 3 Jahren als Web-basierte, virtuelle Bibliothek realisiert. 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 werden viele der hier vorgestellten Konzepte, Verfahren und Ansätze umgesetzt und verwendet.

2.3.3 Meta-Akad als Dienst: Akleon

Die prototypische Implementierung von Meta-Akad wurde im Rahmen von projektspezifischen Praktika weiterentwickelt und vervollständigt und befindet sich unter dem Namen Akleon als Nachweissystem Web-basierter Lehr- und Lernmaterialien als öffentlich zugänglicher Dienst im Produktivbetrieb[1].

2.3.4 Sicherheit durch Einsatz biometrischer Verfahren

Sicherheit ist bei der Dokumentenverwaltung in Datenbanken und bei der Übertragung der Dokumente im Web ein sehr wichtiges Thema. Für die verschiedenen Aspekte der Sicherheit werden heute dazu Verschlüsselungs- und Signaturverfahren eingesetzt. Verschlüsselungsverfahren, also die symmetrische oder die asymmetrische Verschlüsselung, „verbergen" den Inhalt eines Dokumentes, während digitale Signaturverfahren die Echtheit oder Unverfälschbarkeit eines Dokumentes gewährleisten. Letztere berechnen aus einem Text einen eindeutigen Wert. Bei Veränderung des Ursprungstextes ergibt sich bei gleicher Berechnung ein anderer Wert, so dass eine Veränderung unmittelbar entdeckt werden kann.

Im Rahmen einer Diplomarbeit [Kir01] untersuchten wir insbesondere biometrische Verfahren für diese Aufgaben. Als wesentliches Ergebnis stellten wir fest, dass biometrische Messwerte nicht als Schlüssel herangezogen werden können, da sich bei jeder Messung desselben Merkmals ein geringfügig anderer Wert ergibt. Dies hätte zur Folge, dass ein verschlüsselter Text nicht mehr lesbar gemacht werden kann. Biometrische Verfahren können aber zur Zugriffsberechtigung, z. B. auf SmartCards, verwendet werden, weil hierbei die gemessenen Werte durch Schwellwertbildung mit dem hinterlegten Original auf „Gleichheit" verglichen werden. Dadurch entfällt das Merken von Passwörtern, welche ein erhebliches Sicherheitsrisiko bergen (Aufschreiben und Verlieren von Passwörtern, einfach zu erratende Passwörter).

In einer Web-basierten Demonstrationsanwendung wurden verschlüsselte Daten in einer relationalen Datenbank abgelegt und verwaltet. Externe Schlüssel, die den Zugriff und die Dechiffrierung der verschlüsselten Daten erlaubten, wurden in einer SmartCard gespeichert, welche durch ein biometrisches Verfahren gesichert ist. Prinzipiell bietet das untersuchte Verfahren einen ausreichenden

1.) http://www.akleon.de

Schutz vor der Dechiffrierung und Manipulation der Daten, aber keine 100%-ige Sicherheit. Genauso wie die händige Unterschrift mit viel Mühe und Aufwand gefälscht werden kann, kann auch die digitale Signatur gefälscht werden. Allerdings kann die digitale Signatur als ungleich sicherer eingestuft werden, da ihr Fälschungsaufwand sehr viel höher ist. Der breite Einsatz solcher Verfahren ist jedoch momentan noch sehr in Frage zu stellen. Wegen der mangelnden Eingabestabilität der Lesegeräte für biometrische Daten (Fingerabdruck) war der Umgang mit der SmartCard sehr mühsam.

2.3.5 Möglichkeiten der Anwendungsintegration in ORDBS

Objekt-relationale Datenbanksysteme (ORDBS) bieten heute zahlreiche Erweiterungsmöglichkeiten. Über die Erweiterungsschnittstellen ist es möglich, Teile der sonst in der separaten Anwendung vorhandenen Funktionalität in das ORDBS zu integrieren. Mittlerweile können sogar komplette Anwendungsdienste in Form eines Servers im ORDBS ablaufen, um so eine datennahe Verarbeitung zu erzielen. Die Vorteile einer solchen Integration liegen in der Reduktion des Protokoll- und Kommunikationsaufwandes und in der leichteren Konsistenzsicherung zwischen Anwendungs- und Datenbankserver, z. B. bei Einsatz von Pufferungstechniken. Allerdings gibt es bisher wenig Erfahrungen und auch keine Entwurfsregeln für die Nutzung der objekt-relationalen Erweiterungsmöglichkeiten, insbesondere natürlich in Bezug auf die Integration von gesamten Anwendungsdiensten in ein ORDBS. Daher wurde in [FL01] anhand einer an die TPC-W-Spezifikation [TPC00] angelehnten Anwendungsdomäne, einem elektronischen Geschäft (*Web-Shop, E-Shop*), untersucht, ob und welche Vorteile die Verlagerung von Anwendungsfunktionalität in das ORDBS bringen. Die vom TPC-W definierte Testumgebung für Web-basierte Verkaufssysteme wurde deshalb verwendet, weil ein realitätsnahes Szenario und geeignete Bewertungsmaßstäbe vorgegeben werden und zudem bereits erste Testprogramme (Lastgeneratoren) frei verfügbar sind. Der Vorgang der Integration des Anwendungsservers gestaltete sich sehr einfach und erfolgte durch den Austausch des JDBC-Treibers sowie durch das initiale UDF-basierte Starten des Dienstes. Ansonsten wurden keine Änderungen vorgenommen, so dass der Integrationsaufwand als sehr gering bezeichnet werden kann. Es zeigte sich, dass trotz des z. T. doch erst prototypischen Zustands der Erweiterungsschnittstellen des verwendeten ORDBS bei einer Integration des Anwendungsservers im Vergleich zu einem externen Anwendungsserver keine Leistungseinbußen auftreten. Nur bei sehr großer Last waren die derzeitigen Einschränkungen deutlich erkennbar.

3 Individualisierung / Personalisierung

3.1 Portale als Basistechnologie einer Plattform für E-Learning

An die Stelle einer einfachen Web-Präsenz sind Internet-Portale getreten. Beschäftigt man sich mit der Frage nach den zugrunde liegenden Konzepten oder Eigenschaften und beginnt im World-Wide Web mit der Suche, so liefern einschlägige Suchmaschinen über 43 Millionen Treffer für das Schlüsselwort „Portal“. Portale besitzen in der Tat viele unterschiedliche Ausprägungen, die es nicht leichter machen, sie als eigenständiges Konzept zu identifizieren. Eine allgemeine Eigenschaftsbeschreibung kann aber dennoch abgeleitet werden: Portale bilden den Einstiegspunkt für einen gemeinsamen, personalisierten Zugang zu Web-basierten Anwendungen und Informationssystemen (WIS). Die Anforderungen an solche Systeme lassen sich durch zwei Hauptmerkmale charakterisieren: Personalisierung und Integration von Daten bzw. Diensten. Nicht mehr der Anwender selbst muss Daten und Dienste im Internet auffinden, kombinieren und sich aufwändig seine eigene Arbeitsumgebung schaffen, sondern das Portal übernimmt diese Integrations- bzw. Personalisierungsaufgabe. Im Folgenden sollen diese Hauptmerkmale systematisch untersucht werden [IBM00].

3.1.1 Entwicklungsgeschichte

In den Anfängen des WWW war dessen Inhalt und Umfang noch überschaubar. Doch schon sehr bald stiegen Umfang und Anzahl der Web Site nahezu explosionsartig an. Schnell entstanden die ersten Sammlungen von Referenzen („Links“). Thematisch geordnet und manuell verwaltet waren diese so genannten „Directories“ erste Einstiegshilfen, um im WWW verborgene Informationen gezielt zugänglich zu machen. Mit der weiter steigenden Zahl an Datenquellen und der immer höheren Änderungsfrequenz der Daten in diesen Quellen wurde es notwendig, die Verwaltung von statischen Link-Sammlungen durch Indexierungsalgorithmen und dynamische Erzeugung von Link-Sammlungen zu automatisieren [Loe01a].

Web-basierte Informationssysteme

Stichwortsuche und dynamische Präsentation der Ergebnisse konnten durch die Verwendung von Datenbanksystemen (DBS) realisiert werden. Durch die Integration von DBS, die eine effiziente Verwaltung der automatischen Indexierungsergebnisse und eine Web-basierte Suche erst ermöglichten, in das Web und durch Kombination mit redaktionellen Inhalten entstanden die ersten Suchmaschinen. Aber nicht nur der Einsatz von DB-Systemen zur Unterstützung von Suchmaschinen, sondern auch die Anbindung bereits bestehender Datenbanken wurde als eine Anforderung erkannt, um einer breiten Nutzerschicht weltweit den Zugriff auf bestehende Datenquellen über ein einfach zugängliches Medium zu ermöglichen [Loe01a]. Der Aspekt der Integration, bei WIS insbesondere der Datenintegration, ist auch charakteristisch für Portalsysteme, die allerdings weit über die Funktionalität einer bloßen Link-Sammlung hinausgehen.

Integration als ein Hauptmerkmal von Portalen

Eines von zwei Hauptcharakteristika in Bezug auf Portale ist der Aspekt der Integration. Neben der bereits erwähnten Datenintegration, vor allem auch bei Daten verschiedener Quellen, ist die Integration funktionaler Komponenten eine notwendige Eigenschaft dieser Systeme. Sie umfasst nicht nur die Suche über die zugrunde liegenden Daten, sondern auch die Integration von einfachen „Groupware"- oder „Community"-Funktionen [BKL+01], wie z. B. einen Kalender, den Usenet-Zugang oder eine E-Mail-Verwaltung. Integration kann desweiteren die Integration von Anwendungen bzw. Anwendungssystemen beinhalten. Im Umfeld von Unternehmensportalen oder Marktplätzen reicht die Komplexität sogar bis in den Bereich der „Enterprise Application Integration" (EAI).

Mit Personalisierung zum Portal

Das zweite Hauptmerkmal eines Portals ist die Möglichkeit zur Personalisierung. Die angebotenen Daten und Dienste können individuell an die Gewohnheiten und Vorlieben eines Nutzers angepasst werden. Dies geschieht entweder systemseitig, indem beispielsweise das Benutzerverhalten beobachtet wird und personalisierte Inhalte präsentiert werden, ohne dass dem Benutzer bewusst wird, dass Inhalte auf sein spezielles Verhalten hin angepasst wurden. Solches Verhalten ist nicht unüblich: manche Suchmaschinen blenden gezielt Werbeschriften ein, die Bezug auf die eingegebenen Stichwörter nehmen [Smi00]. Kann aber bei einer Portalinteraktion direkt auf eine konkrete Person geschlossen werden, z. B. nach einer Authentifizierung im Bereich Web-basierter Einkaufssysteme, so kann auf Profilinformationen zurückgegriffen und gezielt auf Produkte hingewiesen werden, die aufgrund früherer Bestellungen möglicherweise für den Benutzer interessant erscheinen. Diese „Einkaufshilfen" dienen in der Regel allerdings nur einer zielgruppenorientierten Werbung oder der Verkaufsförderung und sind oft zentrales Element von E-Business-Anwendungen. Es werden Referenzen oder Inhaltsfragmente hervorgehoben bzw. dynamisch hinzugefügt, Verweise auf frühere Bestellungen hergestellt oder Beziehungen zu anderen Kunden gesucht, die ein ähnliches Benutzerverhalten gezeigt haben. Dies erfolgt ohne die Möglichkeit der Einflussnahme seitens der Benutzer.

Daher ist neben der systemseitigen die benutzergesteuerte Personalisierung zu betrachten. Im einfachsten Fall können Teile des entsprechenden Benutzerprofils angepasst werden. Voraussetzung ist allerdings eine Authentifizierung. Eine wirklich umfassende, benutzergesteuerte Personalisierung bedeutet aber, dem Benutzer eine persönliche Sicht auf das entsprechende System zu ermöglichen in der Art, dass Inhalt, Struktur und grafische Präsentation frei veränderbar sind. Eine Anpassung der grafische Präsentation darf sich dabei aber nicht auf die Wahl eines Farbschemas beschränken, sondern muss auch die Möglichkeiten bzw. Einschränkungen des benutzten Endgerätes berücksichtigen, z. B. bei mobilen Endgeräten.

Enterprise Information Portal (EIP)

Die Kombination von Integration und Personalisierung sowie deren umfassender Einsatz machen ein WIS zu einem Portal. Navigationshilfen (Link-Sammlungen), Integration von Inhalten verschiedener Datenquellen, Anwendungs- und Systemintegration sind ebenso elementare Bestandtei-

le wie eine persönliche, selbstgestaltete und deshalb individuelle Systemumgebung. Die umfassendste Form eines Portals stellt wohl das „Enterprise Information Portal“ (oft auch als „Corporate Portal“ oder „Enterprise Portal“ bezeichnet) dar. Die möglichen Komplexitätsstufen zwischen einem einfachen Portal und einem Corporate Portal und deren Klassifikation sind Inhalt des nächsten Abschnitts.

3.1.2 Klassifikation von Portalen

Das aus dem E-Commerce bekannte Klassifikationsschema [MTL99] nach beteiligten Parteien lässt sich auf Portale übertragen. Häufig stellen Portale ohnehin eine Erweiterung von WIS oder E-Commerce-Systemen dar. Deshalb werden solche Systeme oft auch „Commerce Portals“ genannt. Unterscheidet man aber im Allgemeinen nur zwischen „Administration“, „Business“ und „Customer“, so kommt besonders im Zusammenhang mit Mitarbeiterportalen noch die Partei der „Employees“ hinzu. Dem Mitarbeiter („Employee“) kommt tatsächlich eine besondere Rolle zu, denn obwohl er einerseits Eigenschaften eines Kunden („Customer“), also Dienstnutzers besitzt, ist er andererseits der Unternehmung („Business“) zuzuordnen. B2E-Portale, also Mitarbeiter-Portale, bieten ortsunabhängigen Zugang zu Unternehmensdaten und Systemen.

Weitere Portale und deren Klassifikation

Eine weitere Kategorie von Portalen bilden die elektronischen Marktplätze („Trading Hubs“). Zu ihnen zählen nicht nur Auktionssysteme, sondern auch Web-basierte Handelssysteme zur Realisierung unternehmensübergreifender Geschäftsvorgänge, wie z. B. Sales Portals, Procurement Portals im Bereich Beschaffungswesen, Ausschreibungen oder Warenterminbörsen. Suchmaschinen als Zugang zu Angeboten oder Dienstleistungen Dritter mit der Möglichkeit zur Personalisierung werden auch als Mega-Portale bezeichnet. Mega-Portale und Trading Hubs lassen sich auch als B2C- bzw. B2B-Portal kategorisieren, allerdings unterscheiden sie sich stark von diesen in der Anzahl der Nutzer. Im Allgemeinen besteht bei B2B-Portalen eine Geschäftsbeziehung aus jeweils genau einem Beteiligten, während Marktplätze potenziell durch eine Vielzahl von Unternehmen genutzt werden können [Sch01].

Unterscheidet man Portale im Hinblick auf die Zielgruppe, also im Hinblick darauf, welche Interessengruppe angesprochen wird, so gelangt man zu der Unterscheidung von so genannten „Community“-Portalen [BKL+01], „E-Learning“-Portalen[2] [3], Branchenportalen[4], Kundenportalen oder Mitarbeiterportalen. Durch die Fokussierung auf einen bestimmten Interessenbereich spricht man bei einem solchen Portal auch von einem vertikalen Portal (oder auch Vortal).

3.1.3 Funktionale Anforderungen

Die funktionalen Anforderungen an Portalsysteme sind vielschichtig, da die Einsatzmöglichkeiten von Portalen sehr unterschiedlich sind. Eine vollständige Aufzählung aller funktionalen Anforderungen ist daher kaum möglich. Allerdings lassen sich Gemeinsamkeiten in den Funktionalitäten der verschiedenen Portale finden, die typischerweise ein Portal charakterisieren. Einfache allgemei-

2.) http://www.webct.com/
3.) http://www.hyperwave.com/
4.) http://www.ihk.de/

ne Anforderungen ergeben sich bereits aus der grundlegenden Eigenschaft eines Portals als einer Web-basierten Anwendung oder ein Web-basiertes Informationssystem. Darüber hinaus spielt in Bezug auf Portale die Unterstützung der Navigation eine besondere Rolle, d. h. die besondere Unterstützung des Portalnutzers beim Auffinden von Inhalten. Dies umfasst auch die Möglichkeit der Suche. Einem Portalnutzer sollte sowohl eine Volltextsuche als auch eine strukturierte Suche ermöglicht werden. In beiden Fällen sollte die Möglichkeit existieren, Suchkriterien kombinieren zu können. Dabei sollten nicht nur Referenzen auf Ressourcen, sondern vor allen Dingen auch Inhalte gefunden werden. Eine weitere Anforderung ergibt sich unter dem Aspekt der Integration. Integrierte Daten und Dienste sollen sich nahtlos in ein Portal einfügen, so dass beispielsweise keine Notwendigkeit besteht, sich für verschiedene Anwendungen oder Datenquellen einzeln zu authentifizieren (auch „Single Login“ oder „Single SignOn“). Die Verwaltung benutzerspezifischer Zugriffsrechte und somit die Autorisierung ist als Aspekt der Personalisierung zu betrachten. Die funktionalen Anforderungen in Bezug auf diese beiden Aspekte werden im Folgenden untersucht.

Integration

Die Integration mit all ihren Ausprägungen ist eine elementare Eigenschaft eines Portals. Die Datenintegration verschiedener Datenquellen ist zusammen mit einem flexiblen Content-Management das prägnanteste Element eines Portals. Auch die Integration einzelner Funktionen, wie eine oben beschriebene Suche oder eine E-Mail-Nutzung bzw. eine Kalenderfunktion im Groupware-Bereich, ist typisch für Portale, ebenso die Integration von Anwendungen, Anwendungssystemen (ERPs) und von Workflow-Funktionalität zur Unterstützung der Automatisierung von Geschäftsabläufen innerhalb eines Unternehmens (B2E) oder unternehmensübergreifend. Dieser letzte Aspekt der Interoperabilität ist allerdings nur bei speziellen Portalen im Bereich des E-Business besonders ausgeprägt, da selbstverständlich diese Funktionalität gerade dort existenziell wichtig ist.

Personalisierung

Neben den eingangs erwähnten Aspekten der Personalisierung ist das Wissensmanagement elementar für ein Portal. Hierbei ist nicht die bloße Integration von Inhalten bzw. Daten wesentlich, sondern deren Verwaltung in der Form, dass Inhalte aggregiert bzw. kombiniert, kategorisiert oder mit benutzerseitig definierten Anmerkungen[5] ergänzt abgelegt und wieder aufgefunden werden können. Dies kann verknüpft werden mit Groupware-Aspekten, beispielsweise für die Unterstützung von virtuellen Arbeitsumgebungen für Projektteams oder Studentengruppen im Bereich „Web-based Teaching“ (WBT). Hinzu kommt die Anforderung, über eingetretene Änderungen oder dem Erreichen bestimmter Zustände automatisch informiert zu werden (Notifikationsmechanismen).

Die Bereitstellung all dieser Portalfunktionalität geschieht nach [Sch01] konzeptionell durch so genannte horizontale Portale. Im Allgemeinen sind horizontale Portale durch ihre branchenübergreifende Sichtweise charakterisiert, wohingegen vertikale Portale sich durch einen starken Branchenfokus auszeichnen. Sie können aber auch als Verfeinerung oder Spezialisierung eines horizontalen Portals verstanden werden und eine spezifische, interessengruppenorientierte Sicht realisieren.

5.) http://www.w3.org/2001/Annotea/

3.1.4 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen seien mit Hinweis auf [HKM01] hier nur kurz umrissen:

- **Akzeptanz:** Ergonomie, leichte Benutzbarkeit, einfache Installation und Administration,
- **Zukunftssicherheit:** Nutzung von Standards (XML, EJB, Portlets) und offenen Schnittstellen, Erweiterbarkeit, Einsatz von Standardsystemen,
- **Investitionssicherheit:** Skalierbarkeit, Plattformunabhängigkeit,
- **Betriebsicherheit**: technische Datensicherheit, Leistungsgarantien im Hinblick auf Skalierbarkeit und Verfügbarkeit, Sicherheit bei Transaktionen und Zugriffsoperationen.

3.1.5 Eigenschaften unterschiedlicher Personalisierungskonzepte

Im Weiteren soll zunächst nicht näher auf die Integrationsproblematik eingegangen werden, da diese allein nicht portaltypisch ist. Personalisierung hingegen ist viel enger mit der Problematik der Realisierung von Portalen verbunden. Wie eingangs erwähnt, ist dabei zwischen der systemseitigen und der benutzergesteuerte Personalisierung zu unterscheiden.

Systemseitige Personalisierung

Die systemseitige Personalisierung kann ohne Kenntnis des Benutzers geschehen. Der Benutzer hat keinen Einfluss auf die Art der Personalisierung und in vielen Fällen ist es ihm überhaupt nicht bewusst, dass personalisierte Seiten präsentiert werden („user-unaware“). Diese indirekte Art der Personalisierung kann erweitert werden, indem die Möglichkeit geschaffen wird, benutzerseitig Einfluss zu nehmen. Die Personalisierung wird dem Benutzer dadurch natürlich bewusst („user-aware“). Die Personalisierung lässt sich mit Hilfe von Benutzerprofilen realisieren und erfordert in diesem Fall die Speicherung personenbezogener Daten (weshalb es aber aus juristischen Gründen notwendig werden kann, den Benutzer über den Vorgang der Personalisierung in Kenntnis zu setzen). Benutzer können entsprechend ihres Verhaltens aber auch kategorisiert und einer bestimmten Benutzerrolle, d. h. einem gemeinsamen Profil (Anfänger, Profi, Techniker, Dozent, Tutor, Student) zugeordnet werden. Regeln („Business Rules“ [CFP99]) definieren Art und Umfang der Personalisierung [Sch01].

Benutzergesteuerte Personalisierung

Gehen die Möglichkeiten der Personalisierung über die indirekte Einflussnahme auf die systemseitige Personalisierung hinaus, gelangt man zur benutzergesteuerten Personalisierung. Der Benutzer kann aktiv das Erscheinungsbild des Portals beeinflussen. Es können Inhalte selektiert, Farbschemata (so genannte „Themes“ oder „Skins“) ausgewählt oder bestimmte Funktionserweiterungen, wie Kalender, E-Mail o. ä. angepasst werden. In der Regel geschieht dies mit Hilfe so genannter „Portlets“[6][7], die einen definierten Zugriff auf ausgezeichnete Informationen anbieten. Sie repräsentieren Informationskanäle zu verschiedenen Nachrichtenagenturen, Wetterberichten, Dokumenten oder Anwendungen und werden im Allgemeinen als eine Einheit des Zugriffs und der Personalisierung betrachtet, d. h., ein Benutzer kann aus der Vielzahl angebotener Portlets eines Portals die für

6.) http://jakarta.apache.org/jetspeed/site/index.html
7.) http://technet.oracle.com/products/dynamic_services/htdocs/dynamic_services_pdk/pdkds.faq.portlets.and.ds.html

ihn relevante Teilmenge auswählen und auf deren Inhalte zugreifen, wenn er dazu autorisiert ist. Allerdings sind Portlets nichtsdestotrotz vordefinierte Komponenten, die zwar flexibel innerhalb einer gemeinsamen Sicht angezeigt werden können, allerdings keine umfassende Personalisierung erlauben, da sie sich nicht miteinander kombinieren lassen.

Umfassendes, benutzergesteuertes Personalisierungskonzept

Verbunden mit einem umfassenden Personalisierungskonzept ist nicht nur die Anforderung, die zugrunde liegenden Daten benutzerseitig in ihrer Gesamtheit zu betrachten, zu selektieren, zu restrukturieren und individuell zu präsentieren. Weiterhin muss es möglich sein, Funktionen oder Anwendungen mit den Daten zu kombinieren, z. B. einen textuell repräsentierten Termin innerhalb einer Pressemitteilung mit einem Kalendereintrag. Eine bevorzugte Sprache sollte genauso frei wie die grafische Repräsentation gewählt werden können, damit sich unterschiedliche Endgeräte unterstützen lassen (Web-Browser, WAP-Handy, Druckmedien usw.). Diese Eigenschaft wird häufig auch „Multi-Channel Delivery“ (MCD) genannt. Werden dabei zusätzlich auch so genannte „Pervasive Devices“, wie beispielsweise PDAs, unterstützt, so spricht man oft von „Pervasive Portals“.

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 zugrunde liegenden Daten, Struktur und Repräsentation 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].

3.2 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. Die Beschreibung der Anforderungen an eine XML-Anfragesprache [XQL00] formuliert ähnliche Kriterien zur Unterscheidung verschiedener Anwendungsgebiete bzw. Sichtweisen. In beiden Fällen aber werden Daten, im allgemeinsten Sinne, durch eine Menge von XML-Dokumenten repräsentiert. Die Anforderungen an deren Verarbeitung, Verwaltung und Speicherung sind allerdings verschieden. Legt man diese Menge an XML-Dokumenten als Datenbasis im Sinne des deklarativen Web-Site -Managements zugrunde, können hierauf explizit Struktur und auch externe Präsentation beschrieben werden. Durch die Trennung der physischen und logischen Aspekte führt eine Änderung in der internen Repräsentation, z. B. eine einfache Änderung des Dateinamens, nicht zwingend zu einer Änderungen auf der Ebene der externen Präsenation, d. h. zu keiner Änderung in dem *Uniform Resource Identifier* (URI). Ferner lassen sich nun Strukturbeschreibung und Aus-

gabeformat an die Bedürfnisse der Benutzer anpassen, was damit den Anforderungen entspricht, die aus der Personalisierung entstehen. Typischerweise sind im Bereich XML-basierter Web-Anwendungen [BGL+99] eine Menge beliebig strukturierter XML-Dokumente zu verwalten. Der Einsatz von XML erlaubt eine Trennung von Daten und ihrer Repräsentationen. Es sind sogar deskriptive Anfragen [LPV00] an die Menge der XML-Dokumente möglich und auch deren Kombination und Restrukturierung ist vorstellbar, wenn ein jedes XML-Dokument explizit benannt werden kann. Eine Anfrage an die Gesamtheit der zugrunde liegenden Daten ist aber nicht ohne weiteres möglich. Betrachtet man die XML-Dokumente allerdings allein aus einer datenorientierten Perspektive und vernachlässigt ihre physische Repräsentation als textuelles Dokument, so ist es möglich, sie zu einem einzigen logischen Dokument zu integrieren. Diese logische Sicht („*Unified View*“) [Fle01a] auf die Menge der XML-Dokumente erlaubt eine abstrakte Betrachtung der repräsentierten Daten als Grundlage eines umfassenden Personalisierungskonzeptes unter Benutzung von standardisierten Technologien. Diese Sichtweise unterstellt allerdings, dass nur ein geringer Anteil an so genanntem *Mixed Content* [XML00] vorkommt und die Reihenfolge der Elemente innerhalb des Dokuments weniger wichtig ist. Der Entwurf einer Architektur und die zugehörigen Konzepte zur Erstellung des so genannten *Unified View* sollen im Folgenden erläutert werden. Zusammen mit der Heterogenität der zugrunde liegenden Datenquellen ergeben sich folgende Probleme, die in der angegeben Reihenfolge zu überwinden sind:

1. Datenquellen können unterschiedliche Kommunikationsprotokolle verwenden und die zugrunde liegenden Rechnermodelle (Zeichencodierung, Repräsentation von Zahlen) können variieren.
2. Die Daten verschiedener Datenquellen können unterschiedlichen Datenmodelle unterliegen
3. und unterschiedliche Anwendungsschnittstellen (APIs) erfordern.
4. Die Datenquellen können unterschiedliche Möglichkeiten oder Eigenschaften (Source Capabilities) anbieten.
5. Da jede Datenquelle ein eigenes lokales Schema verwendet, muss zunächst eine Möglichkeit gefunden werden, dieses Schema zu extrahieren und zu repräsentieren.
6. Die Menge der einheitlich repräsentierten lokalen Schemata muss integriert werden.
7. Jedes lokale Schema kann eigene Konzepte verwenden, was auf das Problem der semantischen Heterogenität führt.
8. Es können unterschiedliche Anfragesprachen von den Datenquellen eingesetzt bzw. angeboten werden. Allerdings sollte es möglich sein, eine Anfrage in einer gemeinsamen Anfragesprache an die Gesamtheit aller Daten zu spezifizieren.
9. Anfragen sollen zur Definition von Sichten verwendet werden, um eine umfassende Personalisierung zu unterstützen.
10. Es existieren Endgeräte mit unterschiedlichen Eigenschaften.
11. Das Web-basierten Anwendungen immanente Problem der Verfügbarkeit und Netzwerkverzögerungen durch geeignete Sicherungsmaßnahmen berücksichtigt werden.

3.3 Schrittweises Realisierung in SHARX

Bei der Ableitung einer Lösung oder einer Architektur ist beispielsweise die Frage zu betrachten, wie die Dokumentenbasis auf der Ebene der internen Schicht tatsächlich realisiert werden kann. Es erscheint nicht sinnvoll, die gesamten Daten einer Datenquelle durch genau ein XML-Dokument zu repräsentieren und dieses anschließend als Textdokument verarbeiten zu wollen. Vielmehr sollen

Anfragen auf einem die Struktur der Datenquelle beschreibenden Schema beantworten werden. Dabei werden die Anfragen und deren Ergebnisse aber mittels XML beschrieben. Ebenso wenig sinnvoll ist die vollständige Materialisierung aller Daten für die Bereitstellung eines Unified View. Es ist eher eine Komponente vorstellbar, die dynamisch Anfragen gegen den Unified View als globales konzeptionelles Schema beantworten kann, indem Teilanfragen an die jeweiligen Datenquellen propagiert und Einzelergebnisse wieder zusammengeführt werden. Dabei können Strategien zur Vorgenerierung und Datenpufferung unterstützend eingesetzt werden. Dies lässt bereits erahnen, welche komplexen Anforderungen in diesem Zusammenhang an die Realisierung gestellt wird. Die im vorangegangen Abschnitt genannten Punkte werden in SHARX schrittweise angegangen, um die verschiedenen Stufe der Heterogenität bzw. Interoperabilitäts- und Integrationsprobleme zu überwinden und eine Lösung ableiten zu können.

3.3.1 Autonomie und Verteilung

Die Verteilung bzw. Autonomie der Datenquellen erfordert ein dynamisches Integrationskonzept. Die mögliche Materialisierung der Daten ist nur bedingt sinnvoll, findet allerdings in Abschnitt 3.3.11 für die Unterstützung von Caching Beachtung. Um Datenquellen mit unterschiedlichen Kommunikationsprotokollen und verschiedenen Rechnermodellen einheitlich ansprechen zu können, werden Wrapper eingesetzt, die in einem ersten Schritt von den konkreten Systemen und Protokollen abstrahieren. Das entwickelte Wrapper-Konzept unterstützt verschiedene Klassen von Anwendungssystemen und Protokollen, darunter auch entsprechende Netzwerkprotokolle (z. B. Datei/FILE, Web/HTTP, DBS/JDBC, EJB/RMIoverIIOP oder Web Services), um entfernte Aufrufe zu realisieren.

3.3.2 XML-Repräsentation

Die genannten Wrapper führen eine Translation von datenquellenspezifischen Datenmodellen durch und repräsentieren die Daten einheitlich durch die Nutzung von XML. Diese Form der Repräsentation mittels XML ermöglicht die Überwindung struktureller Heterogenität.

3.3.3 Einheitliche AW-Schnittstelle

Im Bereich XML-Daten- oder Anwendungsserver gibt es aber nur sehr wenige Vorschläge für standardisierte Anwendungsschnittstellen (APIs) zum Zugriff auf XML-Dokumente bzw. Daten. XML:DB[8] ist die am weitesten unterstützte API. Sie ist zwar durch die Verwendung der Interface Description Language (IDL) programmiersprachenunabhängig spezifiziert, doch findet sie hauptsächlich im Bereich von Java ihre Anwendung (beispielsweise in Tamino[9] XML Server der Software AG). Die API ist mit nur 13 zu implementierenden Schnittstellen sehr übersichtlich gehalten und kann modular erweitert werden. So existieren Schnittstellen für den Zugriff auf Resource-Objekte (bspw. XML-Dokumente), Collection-Objekte (generische Container), Database-Objekte oder Service-Objekte (Collection -Management, Transaktionen, XPath-Query sowie XUpdate).

8.) http://www.xmldb.org
9.) http://www.tamino.de

3.3.4 Kompensation und Anpassung

Allerdings bieten nicht alle Wrapper eine vollständige Unterstützung aller möglichen Aspekte der der XML:DB-API. Zwei Konzepte zur Kompensation dieser fehlenden Funktionalität wurden in SHARX verwendet. Zum einen können einem Wrapper Adaptoren zugeordnet werden, die auf der Ebene der Datenzugriffsschicht beispielsweise eine fehlende XPath-Anfragemöglichkeit ergänzen. Zum anderen existiert zu jeder Datenquelle bzw. zu ihrem Wrapper eine spezifische Beschreibung ihrer Möglichkeiten (Source Capabilities), so dass auch noch zum Zeitpunkt der Anfrageverarbeitung fehlende Funktionalität in der anfrageausführenden Komponente kompensiert werden kann, indem flexibel anhand der datenquellenspezifischen Beschreibung entschieden wird, ob beispielsweise eine Anfrage an die Datenquellen propagiert werden kann oder ein Dokument vollständig materialisiert wird.

3.3.5 Schemaextraktion und Repräsentation

XML-Dokumente repräsentieren die zugrunde liegenden Datenquellen. Um aber diese zu integrieren und auf ihrer Gesamtheit sinnvolle Anfragen formulieren zu können, ist es notwendig, die jeweiligen Schemata der lokalen Datenquellen zu kennen, d. h. die Struktur der entsprechenden XML-Dokumente. Die bereits genannten datenquellenspezifischen Beschreibung beinhalten auch Informationen über die zugehörigen Schemata (DTDs, XML-Schema-Dokument oder das XML-Dokument). Strukturinformationen werden aus diesen Quellen gewonnen und in einem SHARX-internen Format verwaltet. Diese Struktur ist durch einen kantenannotierten Graph repräsentiert und dient in der weiteren Verarbeitung als Grundlage für die Partitionierung, Integration und Anfrageverarbeitung.

3.3.6 Partitionierung und syntaktische Integration

Auf der Basis des internen Schemas wird die Menge der Dokumente auf Ähnlichkeit untersucht und ähnlich strukturierte Dokumente zu Partitionen zusammengefasst. Ein Element ist ähnlich einem anderen Element genau dann, wenn beide Elemente den gleichen Namen und den gleichen Namensraum verwenden. Partitionen unterteilen die Gesamtmenge aller Dokumente in disjunkte Teilmengen. Diese Teilmengen werden zunächst integriert, um die Grundlage zu bilden, in einem nächsten Schritt den Unified View erstellen zu können. Damit ist eine weitere Abstraktion von der physischen Repräsentation der Daten unternommen, da in diesem ersten Schritt der Integration von der Eigenschaft als physisches XML-Dokument abstrahiert. Hier wird auch die datenorientierte Sichtweise des zugrunde liegenden Konzepts deutlich.

3.3.7 Unified View und semantische Integration

Im vorangegangenen Schritt der Integration wurde anhand der Struktur der Dokumente (Syntax) generisch eine Menge offensichtlich ähnlicher oder sogar gleich strukturierter Dokumente zusammengefasst. Im zweiten Schritt der Integration nun werden die Partitionen unter Berücksichtigung der Bedeutung der Elemente (Semantik) und ihrer Beziehungen zueinander anhand eines vorgegebenem übergeordneten Strukturplans zu dem Unified View kombiniert.

3.3.8 Gemeinsame Anfragesprache

Der Unified View repräsentiert die integrierte Sicht auf die Daten der zugrunde liegenden Datenquellen durch die Verwendung von XML. Es ist daher naheliegend eine XML-geeignete Anfragesprache einzusetzen. Hier bietet sich XQuery an. Diese Anfragesprache bietet die Möglichkeit, Daten zu filtern (Selektion), benötigten Elementen auszuwählen (Projektion) und einen Teil des Unified View zu restrukturieren (Kombination, Aggregation).

3.3.9 Personalisierung

Basierend auf dem Unified View und der Möglichkeit deskriptive Anfragen mittels XQuery zu stellen, ist ein umfassendes Personalisierungskonzept denkbar. Das in SHARX angebotene, umfassende Personalisierungskonzept offeriert darüber hinaus dem Anwender, Beziehungen definieren zu können, die in den Unified View integriert werden, um Annotationen, Verweise oder eigene Konzepte (im Sinne einer Ontologie) zur Unterstützung eigener Abbildungsvorschriften zu definieren (Business Rules). Letzteres wird durch das Konzept der generische Integration erst ermöglicht und der Benutzer kann somit durch die Definition geeigneter Beziehungen zwischen Elementen individuell auf den Integrationsvorgang Einfluss nehmen. Der Grundgedanke von SHARX sieht vor, dass jede Beziehung, die bei der Integration berücksichtigt werden soll, zuvor explizit spezifiziert werden muss.

3.3.10 Customization

In Kombination mit der vorangegangenen Personalisierung ist eine endgerätespezifische Präsentationsform wählbar, da die Daten von der Präsentation getrennt betrachtet werden. Dieses so genannte Multi Channel Delivery (MCD) komplettiert das umfassende Personalisierungskonzept und wird unter zu Hilfenahme von Transformationsvorschriften (XSLT) beschrieben. Transformationsvorschriften, Sichtendefinitionen, Personalisierungsinformationen und MCD-Konfigurationen werden gemeinsam unter einem Benutzerprofil verwaltet.

3.3.11 Verfügbarkeit und QoS

Begleitend zu dem schrittweisen Vorgehen bei der Integration und Auflösen der Heterogenität sind weitere qualitätssichernde Maßnahmen denkbar. Caching kann eingesetzt werden, um das Problem der Verfügbarkeit zu entschärfen. Unterschiedliche Verarbeitungsgranulate auf den verschiedenen Stufen des schrittweisen Vorgehens ermöglichen mehrstufiges Caching. Neben der datenquellenspezifischen Wrapper-internen Caches ist sowohl ein Dokument-Cache als auch ein Ergebnis-Cache denkbar. Der Ergebnis-Cache auf der Ebene des Unified View kann dazu genutzt werden, um den Unified View sukzessive zu materialisieren und Anfragen lokal auszuführen. Dies ist allerdings Fokus der Anfrageoptimierung. Andere Maßnahmen, beispielsweise zur Behandlung von unvollständigen Anfragen, sind denkbar und angedacht, werden aber hier zunächst nicht weiterverfolgt.

3.4 Konzeptioneller Architekturentwurf

Die in Abschnitt 3.3 skizzierten Lösungsanschritte lassen sich direkt in eine übersichtsartige Schichtenarchitektur überführen (Abb. 3.1), welche im Folgenden vorstellt wird:

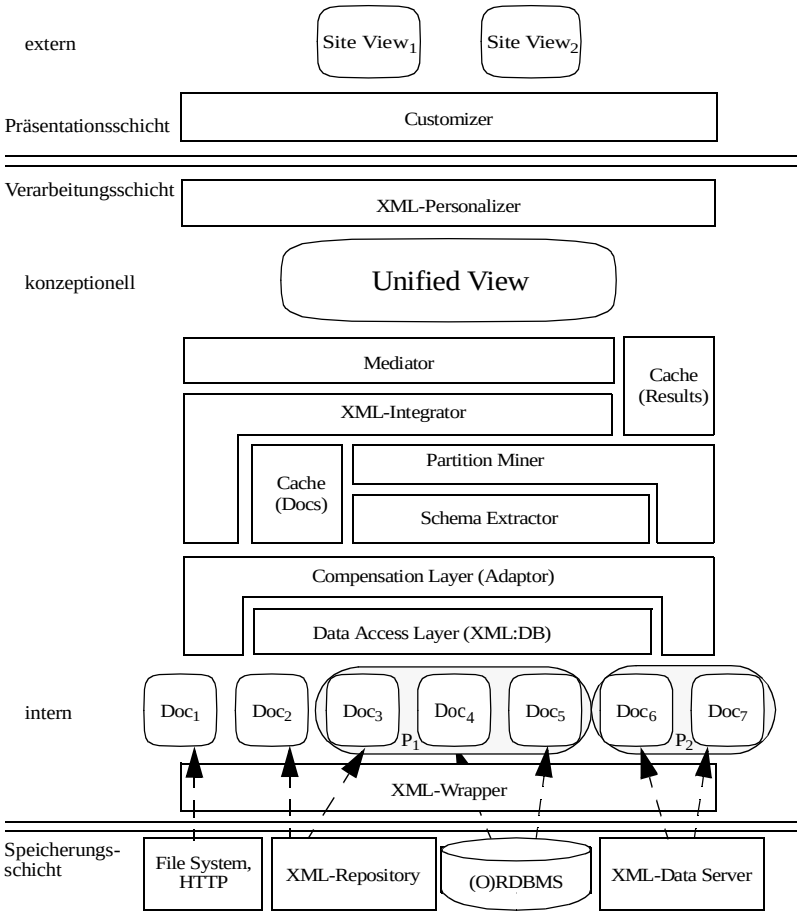


Abb. 3.1: SHARX – Architekturübersicht

Die Architektur ist an eine Mediator-Architektur angelehnt [Wie92]. Der Unified View korrespondiert daher mit einem „Mediated View“, der den Kern der Architektur bildet und um den sich alle anderen Komponenten anordnen. In Abb. 3.1 lässt sich erkennen, dass die Gesamtarchitektur in drei weitere Schichten aufgeteilt ist. Auf der Ebene der untersten Schicht, der Speicherungsschicht, werden die verschiedenen Datenquellen verwaltet. Sie können durch verschiedenste (verteilte) Datenverwaltungssysteme realisiert sein. Auf der Ebene der Präsentationsschicht wird eine endgeräteabhängige Darstellung, die aus den so genannten Site Views abgeleiten wird, dem Benutzer verfügbar gemacht. Die Transformation eines Site View in eine entsprechende externe Präsentationsform kann aus Optimierungsgründen oder zum Zweck der Anonymisierung auch durch Dienste Dritter realisiert werden. Dies schließt natürlich auch die Möglichkeit mit ein, die Transformation durch einen geeigneten Web-Browser (oder User Agent) ausführen zu lassen.

Der tatsächliche Kern der Architektur aber liegt in der mittleren Schicht. Diese für die eigentliche Verarbeitung verantwortliche Schicht ist ihrerseits wieder in drei Teile gegliedert. Die Verarbeitungsschicht orientiert sich dabei an dem Drei-Schichten-Modell des Datenbankentwurfs und reali-

siert die für der Verarbeitung bzw. Abstraktion verschiedenen notwendigen Schritte: Trennung von physischen und logischen Aspekten der Datenrepräsentation, Erzeugung des Unified View und die Unterstützung externer Sichten für die Selektion, Projektion und Restrukturierung der Daten.

Die Verarbeitungsschicht teilt sich somit in die folgenden drei Schichten: intern, konzeptionell, extern. Auf der Ebene der internen Schicht erfolgt der erste Schritt der Abstraktion von der physischen Repräsentation der Daten, die in der Speicherungsschicht verwaltet werden, durch die ausschließliche Nutzung von XML. Die mögliche Heterogenität der verschiedenen Datenquellen wird durch die Verwendung eines einheitlichen Modells zur Repräsentation der Daten, entsprechend eines einheitlichen Datenmodells, beherrschbar. Allerdings ist die interne Schicht geprägt durch die Repräsentation der Daten durch Dokumente, wodurch Anfragen an die Gesamtheit der Daten nur sehr bedingt möglich sind. Erst durch einen zweiten Abstraktionsschritt, in dem die XML-Dokumente unter einer datenorientierten Sichtweise zu dem Unified View kombiniert werden, wird vollständig von der physischen Repräsentation der Daten abstrahiert, da die Dokumenteneigenschaft der XML-Dokumente aufgelöst wird. Basierend auf dem Unified View lassen sich nun deklarativ externe Sichten definieren, die als Grundlage einer umfassenden Personalisierung dienen können. Restrukturierung und damit letztlich auch die Kombination von Daten verschiedener Datenquellen ist möglich.

3.4.1 Eigenschaften des Unified View

Dem Unified View liegt die Beobachtung zugrunde, dass Mengen von XML-Dokumenten bestimmt werden können, deren Elemente jeweils eine ähnliche Struktur aufweisen. Dokumente ähnlicher Struktur werden durch den „Partition Miner“ zu jeweils einer Partition zusammengefasst und kombiniert. In einem weiteren Schritt werden diese so entstandenen Partitionen zu dem Unified View vervollständigt („XML-Integrator“). Der „XML-Personalizer“ ist für die Generierung der externen personalisierten Sichten verantwortlich. Durch die Verwendung von Referenzen mittels XLink [XLi01] werden Beziehungen zwischen Elementen explizit modelliert und können sowohl für die Navigation als auch für die Restrukturierung und die Verwaltung von Anmerkungen benutzt werden. Dabei lassen sich potenziell alle Formen von XLinks unterstützen, insbesondere so genannte „Third-Party“-Links, die zwar extern definiert, aber ebenso wie alle zugrunde liegenden XML-Dokumente in den Unified View integriert werden. Die folgenden Absätze beschreiben nun den Weg von der Dokumentenbasis zu dem Unified View.

3.4.2 Dokumentenbasis

Die Dokumentenbasis wird einzig durch die Gesamtheit der zugrunde liegenden XML-Dokumente gebildet. Ihre Elemente können nach funktionalen Kriterien klassifiziert werden. In unserem Kontext haben wir fünf Klassen identifiziert:

- operational
- strukturierend
- navigationsbezogen
- funktional
- organisatorisch.

Die erste Klasse beschreibt die Dokumente mit den eigentlichen operationalen Daten (Publikationslisten oder Sammlung von Web-Favoriten bzw. Bookmarks). Die strukturierenden Dokumente beeinflussen indirekt den Mechanismus der Erzeugung des Unified View. Sie enthalten verschachtelte leere XML-Elemente, um gerüstartig die Struktur der zu verwaltenden Web-Site zu definieren. Explizite Links [XLi01] zwischen Dokumenten finden sich in den navigationsbezogenen Dokumenten. Sie drücken Abhängigkeiten zwischen diesen Dokumenten aus, die zum Zweck der Navigation verfolgt oder zur Restrukturierung (Komposition oder Aggregation) benutzt werden können. Funktionale Dokumente spezifizieren eine einfache Applikationslogik, z. B. die für die Definition dynamischen Inhalts basierend auf Aufrufen externer Systeme mittels *SOAP*. Die letzte Klasse der organisatorischen Dokumente beinhaltet benutzerbezogene Daten wie Zugriffskontrollinformationen („Access Control Lists“). Zu dieser Klasse gehören auch Dokumente, die nutzungsbezogene Daten (z. B. Profile) beinhalten, um so genannte „Business Rules“ zu unterstützen. Sie dienen somit der Personalisierung und können beispielsweise auch zielgruppenorientierte Werbung ermöglichen.

3.4.3 Erzeugung des Unified View

Die Trennung von Partitionierung und Integration der Dokumente bzw. der Daten ergibt sich aus dem besonderen Vorgehen bei der Erzeugung des Unified View. Die Dokumente werden dazu aus einer datenorientierten Perspektive betrachtet. Es ist durchaus einsichtig, dass Ähnlichkeiten zwischen Dokumenten durch die Betrachtung des jeweiligen Inhaltsmodels („Content Model“ [XML00]) gefunden werden können. Ähnliche Dokumente werden derart zu einer Partition zusammengefasst, dass zunächst, falls vorhanden, die DTD analysiert wird und die Elementnamen zusammen mit ihrem zugehörigen Namensraum („Namespace“ [XNS99]) ausgewertet werden (z. B. Publikationslisten, RDF-Daten oder Adressinformationen). Ein Dokument kann dabei nur zu genau einer Partition gehören, deren jeweilige Elemente in einem weiteren Schritt erst gruppiert und dann kombiniert werden. Nach der Kombination der die operationalen Daten beinhaltenden Dokumente werden sie zu dem Unified View integriert, indem sie in der entsprechenden Stelle des Gerüsts eingefügt werden, das durch die strukturierenden Dokumente definiert wird. Die strukturierenden Dokumente bilden wiederum eine eigene Partition und werden auf die gleiche Art und Weise kombiniert wie die operationalen Dokumente. Bisher werden aber nur strukturelle Eigenschaften betrachtet. Die Ähnlichkeit zweier Elemente haben wir auf die Ähnlichkeit des jeweiligen Inhaltsmodells zurückgeführt, also auf die Identität bezogen auf die DTDs, Namensräume oder Elementnamen reduziert. Semantische Ähnlichkeiten zwischen Elementen unterschiedlicher Inhaltsmodelle werden zunächst nicht betrachtet.

Durch den Einsatz der navigationsbezogenen Dokumente kann aber wiederum eine explizite semantische Beziehung zwischen Elementen bzw. Dokumenten ausgedrückt werden. Diese Beziehungen können zum Zweck der Navigation oder für die Restrukturierung eingesetzt werden, indem nicht die Referenz, sondern der jeweilige referenzierte Teilbaum („Element Tree Fragment“ [XML00]) in die entsprechende Stelle des Unified View eingesetzt wird. Eine derartige Behandlung einer Referenz entspräche einer Verbundoperation („Join“) aus dem Bereich der Datenbanksysteme.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE PUBLICATIONS
        SYSTEM "file:///xml/dtd/pubs.dtd">
<PUBLICATIONS>
   <PUBLICATION YEAR="2000">
        <TITLE>t1</TITLE>
        <AUTHOR>a1<AUTHOR>
        <AUTHOR>a2<AUTHOR>
        <CONFERENCE>...</CONFERENCE>
        <ABSTRACT HREF="http://external.net/rs1"/>
        <DOWNLOAD HREF="http://external.net/rs2"/>
   </PUBLICATION>
<PUBLICATIONS>
```

Dokument 1

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE PUBLICATIONS
        SYSTEM "file:///xml/dtd/pubs.dtd">
<PUBLICATIONS>
   <PUBLICATION YEAR="2000">
        <TITLE>t2</TITLE>
        <AUTHOR>a1<AUTHOR>
        <CONFERENCE>...</CONFERENCE>
        <ABSTRACT HREF="http://external.net/rs3"/>
        <DOWNLOAD HREF="http://external.net/rs4"/>
   </PUBLICATION>
<PUBLICATIONS>
```

Dokument 2

Abb. 3.2: Beispiel zweier operationaler Dokumente

Navigationsbezogene Dokumente können konzeptionell eine Vielzahl von verschiedenartigen Referenztypen beinhalten: „Outbound Links“, „Inbound Links“, „Third-Party Links“ oder „Link Bases“ [XLi01]. Diese Referenztypen sind relativ einfach in den Unified View integrierbar, da sie alle in eine äquivalente Referenz vom Typ „Outbound Links“ transformiert werden können. Funktionale und organisatorische Dokumente werden ähnlich wie operationale Dokumente behandelt, allerdings werden sie ausgewertet, bevor sie integriert werden. Der Vorgang der Integration sollte weitestgehend generisch sein.

3.4.4 Beispiel

Im Folgenden sollen die Konzepte der Erzeugung des Unified View anhand eines Beispiels illustriert werden. In Abb. 3.2 sind aus diesem Grund zwei operationale Dokumente angegeben. Sie beschreiben Publikationen, enthalten keinen Mixed Content und ihr Inhaltsmodell ist in einer extern gespeicherten DTD beschrieben. Aufgrund ihres ähnlichen Inhaltsmodells werden sie derselben Partition zugeordnet und gruppiert (Abb. 3.3). Es wird auch eine Partition strukturierender Dokumente gebildet (Abb. 3.4), deren Elemente gerüstartig die Struktur des Unified View definieren. Zusammen mit den gruppierten Dokumenten der jeweiligen Partitionen werden sie zu dem Unified View integriert (Abb. 3.5).

```
<PUBLICATIONS>
  <PUBLICATION YEAR="2000">
      <TITLE>t1</TITLE>
      <AUTHOR>a1<AUTHOR>
      <AUTHOR>a2<AUTHOR>
      <CONFERENCE>...</CONFERENCE>
      <ABSTRACT HREF="http://external.net/rs1"/>
      <DOWNLOAD HREF="http://external.net/rs2"/>
  </PUBLICATION>
  <PUBLICATION YEAR="2000">
      <TITLE>t2</TITLE>
      <AUTHOR>a1<AUTHOR>
      <CONFERENCE>...</CONFERENCE>
      <ABSTRACT HREF="http://external.net/rs3"/>
      <DOWNLOAD HREF="http://external.net/rs4"/>
  </PUBLICATION>
<PUBLICATIONS>
```

Abb. 3.3: Gruppierte Partition der Dokumente 1 und 2

Ausnutzen von Namensräumen

Dokumente, die nicht den gleichen Dokumenttyp besitzen, können unter Umständen auch einer Partition zugeordnet werden, wenn der Namenraum und der jeweilige Elementname entsprechend analysiert wird. Besitzen zwei Dokumente nicht das gleiche Wurzelelement, aber lassen sich Teilbäume mit Elementen gleichen Namens und Namensraums identifizieren, so können auch diese Dokumente einer Partition zugeordnet werden. Beispielsweise könnte ein Dokument in der Dokumentenbasis vorkommen, das die gleichen Elemente benutzt wie Dokument 1 aus Abb. 3.2, aber nicht vom Typ <PUBLICATIONS> ist.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE SITEGRAPH
      SYSTEM "file:///xml/dtd/sitestructure.dtd">
<SITEGRAPH>
  <STAFF>
      <MEMBER/>
  </STAFF>
  <PUBLICATIONS/>
  <JOBS/>
  <PROJECTS/>
</SITEGRAPH>
```

Dokument 3

Abb. 3.4: Strukturierendes Dokument

Mit einer geeigneten Namensraumdefinition, die auf alle drei Dokumente anzuwenden ist, wäre es möglich, das Dokument ohne spezifiziertes Inhaltsmodell derselben Partition wie Dokument 1 oder 2 zuzuordnen und es entsprechend zu gruppieren und zu kombinieren. Kann nicht auf eine Namensraumdefinition zurückgegriffen werden, so ist allein der Elementname zu betrachten.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SITEGRAPH>
  <STAFF>
    <MEMBER/>
  </STAFF>
  <PUBLICATIONS>
    <PUBLICATION YEAR="2000">
      <TITLE>t1</TITLE>
      <AUTHOR>a1<AUTHOR>
      <AUTHOR>a2<AUTHOR>
      <CONFERENCE>...</CONFERENCE>
      <ABSTRACT HREF="http://external.net/rs1"/>
      <DOWNLOAD HREF="http://external.net/rs2"/>
    </PUBLICATION>
    <PUBLICATION YEAR="2000">
      <TITLE>t2</TITLE>
      <AUTHOR>a1<AUTHOR>
      <CONFERENCE>...</CONFERENCE>
      <ABSTRACT HREF="http://external.net/rs3"/>
      <DOWNLOAD HREF="http://external.net/rs4"/>
    </PUBLICATION>
  <PUBLICATIONS>
  <JOBS/>
  <PROJECTS/>
</SITEGRAPH>
```

Abb. 3.5: Unified View für Dokument 1, 2 und 3

Semantische Beziehungen und Personalisierung

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SITEGRAPH>
  <STAFF>
    <MEMBER>
      <NAME>a1</NAME>
      <PUBLICATIONS>
        <PUBLICATION YEAR="2000">
          <TITLE>t1</TITLE>
          <TITLE>t2</TITLE>
      </PUBLICATION>
    </MEMBER>
    ...
  </STAFF>
  <PUBLICATIONS>
    ...
  </PUBLICATIONS>
  <JOBS/>
  <PROJECTS/>
</SITEGRAPH>
```

Abb. 3.6: Eine externe Sicht auf Dokument 1, 2 und 3

Die Identifikation semantischer Beziehungen zwischen Elementen, beispielsweise zwischen Autoren und Mitarbeitern, ist nicht Teil dieses generischen Integrationsprozesses. Vielmehr sind solche Informationen durch den Einsatz von so genannten navigationsbezogenen Dokumenten explizit zu spezifizieren. Der Algorithmus ist dann in der Lage, diese bei der Integration zu berücksichtigen, was den Vorteil besitzt, dass solche Informationen auch zu einem Teil der Daten werden. Die Identifikation bzw. Spezifikation semantischer Beziehungen kann sowohl manuell als auch (semi-)automatisch durch Verwendung entsprechender Algorithmen des Web Mining, Einsatz von KI-Methoden oder CBR-Techniken usw. geschehen. Da durch den Unified View eine logische Sicht auf die Gesamtheit der Dokumentenbasis gegeben wird und dem Benutzer explizite Informationen über die Beziehung zwischen Elementen bereitgestellt werden, ist nun eine

umfassende Personalisierung vorstellbar. Das Filtern von Daten (Selektion), die Auswahl von benötigten Elementen (Projektion), das Restrukturieren (Kombination, Aggregation) eines Teils des Unified View und letztlich auch die Anpassung der externen grafischen Präsentation wird möglich. Beispielsweise können Autoren und Mitarbeiter aus dem zuvor genannten Beispiel wie in Abb. 3.6 neu kombiniert werden.

3.5 Systementwurf

Der zuvor in Abb. 3.1, „SHARX – Architekturübersicht," auf Seite 18 skizzierte Entwurf orientiert sich an einer Drei-Schichten-Architektur, deren mittlere Schicht in drei weitere Schichten unterteilt ist. Unberücksichtigt blieb in Abb. 3.1 zunächst noch die Verwaltung der Registrierungsdaten in einem Metadaten-Repositorium, die unterstützenden Komponenten der Anfrage- und Personalisierungsverarbeitung und die Steuerung und Überwachung der internen Abläufe durch eine gemeinsame Choreography-Komponente. In Abb. 3.7 ist die vollständige SHARX-Komponentenarchitektur dargestellt. Analog zu der Architekturübersicht bilden auch hier Datenanbindung, Integration, Anfrage und Personalisierung den eigentlichen Kern der Architektur.

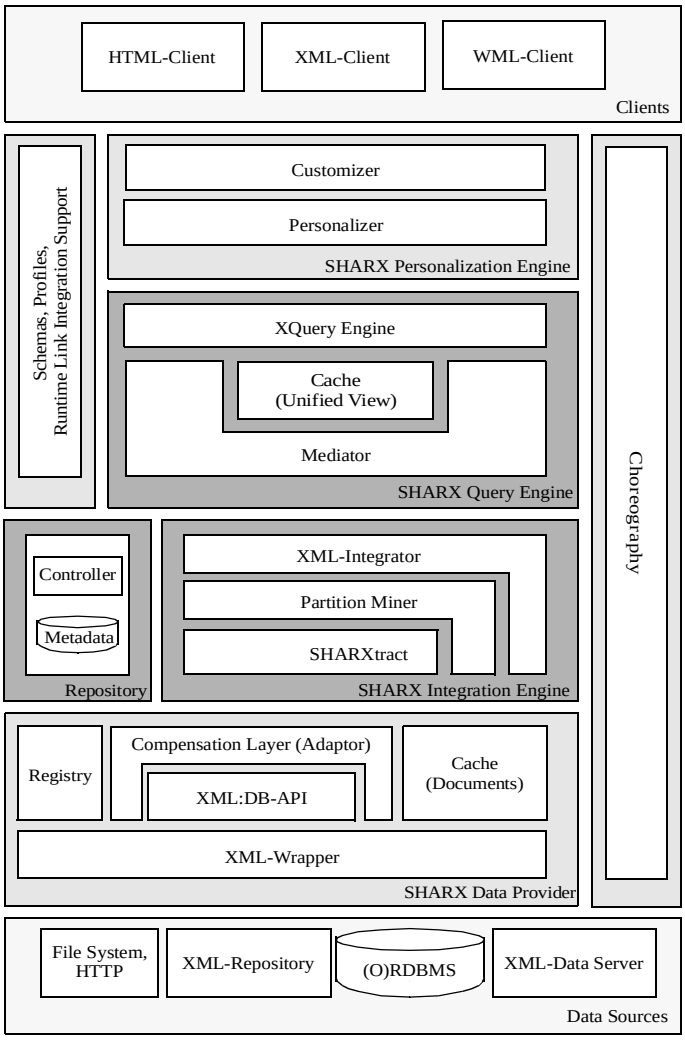


Abb. 3.7: SHARX – Komponentensicht

Im Gegensatz zum vorherigen Entwurf sind Komponenten ergänzt und zu logischen Einheiten zusammengefasst worden. Die Basis bildet der so genannte Data Provider. Hier sind XML:DB-verwendende Wrapper, kompensierende Komponenten (Adaptoren) und der Dokumenten-Cache wiederzufinden. Darüber befindet sich die Integration Engine, die Komponenten zur Behandlung der statischen Integrationsaspekte beinhaltet. Komponenten zur Schemaextraktion (SHARXtract), Analyse (Partition Miner) und Integration sind hier zusammengefasst. Ein Repositorium zur Speicherung der extrahierten Schemata und Integrationsinformationen ist ergänzt worden.

Die Query Engine wird durch den Mediator, den Anfrageprozessor (XQuery Engine) und den Unified View gebildet. Die Unified-View-Komponente kann auch Cache-Funktionalität beinhalten. Aus der Sichtweise der Systemarchitektur ist der Unified View eine der Verarbeitung zugrunde liegende Datenstruk-

tur, die Zwischenergebnisse vorhalten kann und letztlich die Basis für die Ausführung einer Anfrage darstellt. Personalisierung und endgerätespezifische Anpassung und Transformation (Customization) sind in der Personalization Engine zusammengefasst und repräsentieren die Server-seitige Präsenationsschicht. Data Provider, Integration Engine, Query Engine und Personalization Engine bauen sukzessive aufeinander auf. Schichtübergreifende Funktionalität wird durch zwei weitere Komponenten bereitgestellte. Die Choreography-Komponente (auch: Notification Back-End) übernimmt die Rolle der Steuerung und Überwachung der internen Abläufe. Bei Registrierung, Änderung oder Löschung eines aus einer Datenquelle zu integrierenden Dokuments, werden entsprechende Mitteilungen (Notification) erzeugt und durch die Choreography-Komponente weitergeleitet, d. h. entsprechende Aktionen (beispielsweise Cache-Invalidierung oder Neupartitionierung) werden angestoßen. Die zweite schichtübergreifende Komponente unterstützt die Integration von Beziehungen und Personalisierung während der Anfrageverarbeitung und stellt den oberen Schichten Schemabeschreibungen in Form von XML-Schema-Dokumenten zur Verfügung, die aus der internen Schemabeschreibung generiert werden.

3.6 Anbindung der Datenquellen

In Mediator- bzw. Integrationssystemen übernehmen Wrapper-Komponenten Schematranslation und -Transformation sowie Anfrage oder Extraktion von Daten aus entfernten, verteilten, heterogenen Datenquellen. Diese Aufgabe können generische, generierte oder spezialisierte, eigens für genau eine Datenquelle realisierte Komponenten übernehmen. Generische Wrapper zeichnen sich dadurch aus, dass sie aufgrund einer Beschreibung der Eigenschaften und Möglichkeiten einer Quelle für eine ganze Klasse von Datenquellen verwendbar sind. Mit der Verwendung von XML konnten Wrapper und entsprechende Systeme sehr viel universeller gestaltet und eingesetzt werden. Algorithmen bzw. Realisierungen für die Generierung von XML aus relationalen Daten oder für die Datenextraktion aus Web-Ressourcen können leicht angepasst und in SHARX integriert werden. Das Web stellt eine Vielzahl an Listen über verschiedenste Wrapper-Werkzeuge zur Verfügung[10].

3.6.1 Flexible Kopplung durch XML-Wrapper in SHARX

Datenquellen sind konzeptionell heterogen in ihrer Struktur und Semantik. Sie können unterschiedliche Datenmodelle für die Repräsentation ihrer Daten benutzen. Als ein einheitliches Modell für die Repräsentation von Daten verschiedener Datenmodelle bietet sich XML an. Die Familie der XML-Technologien ist allerdings groß. Für den Zugriff auf XML-Dokumente definiert das World Wide Web Consortium (W3C) das XML Information Set [XIS01] als eine abstrakte baumbasierte Darstellung von XML und spezifiziert mit dem DOM [DOM02] Operationen auf dieser Struktur.

Neben der DOM-Schnittstelle ist mit SAX eine weitere Schnittstelle zum Zugriff auf XML-Dokumente entstanden, die einen ereignisbasierten Ansatz verkörpert. Dies ermöglicht die Verarbeitung (Parsing) eines Dokuments in kleinen Einheiten (Events) ohne die Notwendigkeit, das gesamte Dokument, beispielsweise im Hauptspeicher, zu materialisieren. Allerdings können beide Schnittstellen für die Verarbeitung von XML benutzt werden; sie besitzen aber beide auch spezifische Vor- und Nachteile.

10.) http://www.wifo.uni-mannheim.de/~kuhlins/wrappertools/
http://www.gi-ev.de/informatik/lexikon/inf-lex-xml-strukturakquisition.shtml

Ferner existiert eine Vielzahl von Implementierungen, die auf bestimmte Einsatzbereiche hin optimiert worden sind. Die „Java API for XML Processing“ (JAXP) [JAXP02] benutzt einen so genannten Plugability Layer, um flexibel verschiedene Schnittstellen und Implementierungen einzusetzen bzw. zu kombinieren. Der Plugability Layer bildet dabei eine einheitliche Schnittstelle zu den jeweils eingesetzten Komponenten (Abb. 3.8.). Diese Idee des Plugability Layer wurde bei der Realisierung von SHARX übernommen, um verschiedene XML-Zugriffschnittstellen (DOM, SAX, Text) zu unterstützen und ein Rahmenkonzept (XML:DB[11]) für die Architektur der XML-Wrapper des Data Provider vorzugeben. Die besonderen Eigenschaften und Möglichkeiten einer Wrapper-Implementierung bzw. der zu integrierenden Datenquelle werden dem Kernsystem über eine geeignete Metadatenschnittstelle bekannt gemacht. Eigenschaften beschreiben beispielsweise die Möglichkeit, Daten ändern, anfragen oder vollständig materialisieren zu können.

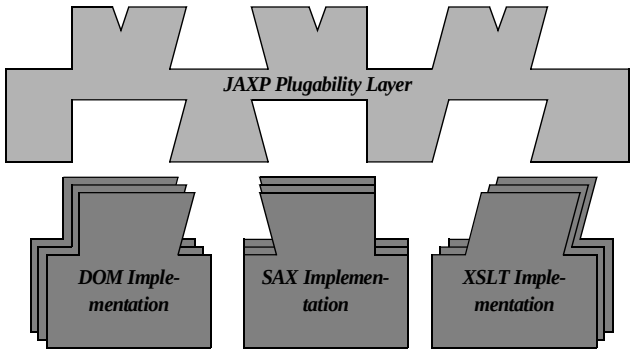


Abb. 3.8: JAXP Plugability Layer

In Abb. 3.9 ist schematisch dargestellt, wie die Architektur eines XML-Wrapper sich gestaltet, wenn die zuvor genannten Anforderungen erfüllt werden sollen. Man erkennt links das SHARX-Kernsystem. In der Mitte ist die eigentliche Wrapper-Komponente und rechts die Datenquelle abgebildet. Die Wrapper-Komponente unterteilt sich in einen Anwendungsteil, der einen starken Framework-Charakter besitzt, also auf dem Konzept des Plugability Layer aufbaut und strikten Vorgaben folgt, und in einen datenquellenspezifischen Teil. Die datenquellenspezifische Subkomponente ist stark an den Eigenschaften der Datenquelle orientiert und realisiert die nicht unbedingt XML-bezogenen Aspekte der Komponente, z. B. ist in dem Fall relationaler Datenbanksysteme die datenquellenspezifische Subkomponente in Java durch den JDBC-Treiber realisiert. Befinden sich nun alle Komponenten auf genau einem System, können alle Funktionsaufrufe lokal behandelt werden. Allerdings gibt es drei weitere Fälle, wenn mindestens der Zugriff auf eine Komponente einen entfernten Zugriff benötigt:

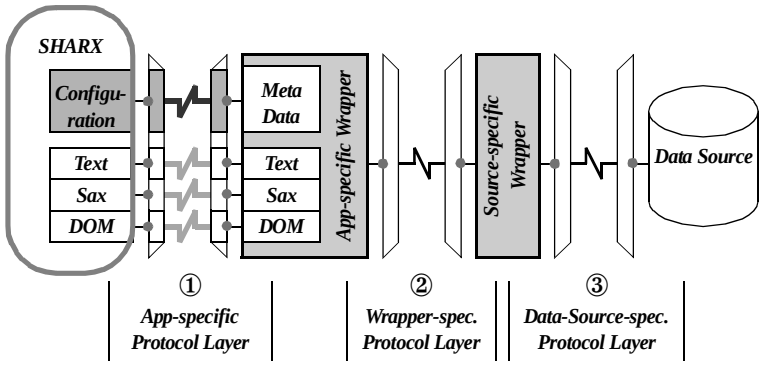


Abb. 3.9: XML-Wrapper-Architektur

- Entfernter Zugriff auf XML-Wrapper (1)
- Verteilter Wrapper (2)
- Entfernter Datenquellenzugriff (3).

11.) Die Spezifikation der XQuery API for Java ist zwar bereits im Entstehen, allerdings momentan noch in einem sehr frühen Stadium, so dass hier von einer Verwendung abgesehen werden musste!

Der Fall des entfernten Datenquellenzugriffs tritt ein, wenn beispielsweise Web-Ressourcen (beispielsweise über HTTP) oder Datenbanksysteme (beispielsweise durch JDBC) anzubinden sind. Besitzt die Datenquelle selbst nicht die Möglichkeit der Benutzung eines Netzprotokolls, kann ein Teil des Wrapper lokal bei der Datenquelle angesiedelt sein. Dies kann auch der besseren Skalierbarkeit dienen, da die beiden Subkomponenten auf unterschiedlichen Maschinen eingesetzt werden können. Weiterhin ist es denkbar, auch seitens SHARX über entfernte Aufrufe mit der anwendungsspezifischen Subkomponente zu kommunizieren. Um an den Stellen 1, 2 und 3 sowohl lokale als auch entfernte Aufrufe zu unterstützen, können dort drei Protokollschichten identifiziert werden:

- Anwendungsspezifische Protokollschicht (1)
- Wrapper-spezifische Protokollschicht (2)
- datenquellenspezifische Protokollschicht (3).

Die Unterstützung eines anwendungsspezifischen Protokolls geschieht in SHARX u. a. über so genannte Adaptoren der Kompensationsschicht, wenn erforderlich auch für entfernte Zugriffe. Die Funktionen und Aufgaben der Adaptoren werden in Abschnitt 3.6.2 näher erläutert. Die konkrete Realisierung des anwendungsspezifischen Wrapper stellt in SHARX eine Implementierung der XML:DB-Schnittstelle dar, die beispielsweise auf der Wrapper-spezifische Protokollschicht JDBC verwenden. Die datenquellenspezifische Wrapper-Implementierung ist dann folglich ein herstellerspezifischer JDBC-Treiber, der zur Kommunikation mit dem Datenbanksystem auf der Ebene der atenquellenspezifischen Protokollschicht ein natives DBS-Protokoll verwenden kann. Die Datenquelle ist in diesem Beispiel ein (objekt-)relationales Datenbanksystem.

In SHARX liegt aber das Hauptaugenmerk nicht auf der Implementierung oder Generierung von Wrapper-Komponenten, sondern SHARX beschäftigt sich mit den konkreten Problemen bei der Integration von XML-Dokumenten unter besonderer Berücksichtigung von Beziehungen und den Aspekten der Personalisierung. Wrapper dienen hier zunächst einmal nur der Abstraktion von den jeweiligen Datenquellen und ihrer spezifischen Datenmodellen und dienen letztlich allein der Erzeugung von XML aus beliebigen Datenverwaltungssystemen. Allerdings bietet das zu verwendete Rahmenkonzept (XML:DB) einige interessante Möglichkeiten der Variation bei der Anfrageverarbeitung. Es ist möglich z. B. unter Berücksichtigung der Größe eines XML-Dokuments die Zugriffsschnittstelle zur Laufzeit flexibel zu wählen, d. h., kann oder soll ein Dokument bzw. Anfrageergebnis aufgrund der Größe nicht vollständig materialisiert werden, stellt vielleicht SAX die optimale Zugriffsschnittstelle dar, wohingegen die textbasierte Schnittstelle günstiger sein kann, wenn das Dokument in der Query Engine vollständig materialisiert werden soll. Kriterien für die Entscheidungsfindung ergeben sich durch die Eigenschaften und Möglichkeiten der Datenquellen sowie aus dem Laufzeitverhalten (z. B. Statistiken) und den Anfragen selber. SHARX bietet die Möglichkeit, eigene Wrapper-Implementierungen, SHARX-spezifische (dateisystembasiert, HTTP-basiert, JDBC-basiert) und herstellerspezifische (Tamino oder Xindice) XML-Wrapper einzusetzen.

3.6.2 Flexible Anpassung durch XML-Adaptoren in SHARX

Auf die jeweiligen XML-Wrapper wird allerdings durch SHARX nicht direkt zugegriffen, sondern der Zugriff erfolgt über eine Proxy-Schicht, die der Kompensation seitens der Wrapper nicht zur Verfügung gestellter Funktionalität dient. Ferner ist über diese so genannten Adaptoren mehr Kontrolle beim Zugriff auf die jeweiligen Datenquellen möglich: Spezifische Autorisierung, Ressourcen-Management, Pooling oder Caching wird möglich. Durch das Plugability-Layer-Konzept kann die Menge der XML-Wrapper und Adaptoren beliebig erweitert und flexibel kombiniert werden. Unterstützt werden momentan folgende Adaptoren:

- Native (Identität)
- Local
- Internal
- Cache mit XPath-Unterstützung.

Im Zuge der Datenquellenanbindung werden die zu verwaltenden Dokumente in einem Repositorium mit entsprechenden Metadaten, also einer Beschreibung der besonderen Eigenschaften eines Dokuments bzw. des zugrunde liegenden Datenverwaltungssystems, registriert. Der native Adaptor verwendet diese Daten allerdings nicht, sondern reicht alle Anforderungen unverändert weiter (Identität) und informiert die Choreography-Komponente über eventuelle Änderungen. Hingegen verwendet der lokal arbeitende Adaptor die im Repositorium verwaltenden Metadaten auf der Ebene der Kollektionen für die Unterstützung der Navigation auf diesen. Kollektionen können hierarchisch organisiert sein und somit wieder aus Kollektionen und Dokumenten bestehen. Der lokal arbeitende Adaptor fragt nun nicht das Zielsystem an, um die Navigation über die Kollektionen zu einem Dokument zu realisieren, sondern erzeugt nur beim Zugriff auf eine registrierte Ressource einen Zugriff auf das Datenverwaltungssystem und verwendet ansonsten die Daten des Repositoriums. Dies hat den Vorteil, dass keine Kollektionen oder Ressourcen angefragt werden können, die nicht registriert sind und für die somit keinerlei Strukturen bekannt sind. Ferner ist es nicht erforderlich, eventuell entfernte Aufrufe an das Zielsystem durchzuführen. Auch kann so das Repositorium selbst durchsucht werden. Der Internal-Adaptor wird eigentlich wie ein XML-Wrapper verwendet und eingesetzt, allerdings stellt auch er einen Proxy zu einer konfigurierbaren internen Implementierung dar, die als lokale Datenquelle für strukturierende und navigationsbezogene Dokumente dient. Auch der Cache kann diese lokale Datenquelle nutzten. Dann besteht der einzige Unterschied zwischen der Cache- und der Internal-Variante in der Art der Adressierung. Der Cache ist transparent für den externen Zugriff, wohingegen im Falle der Internal-Variante aus der Adressierung explizit die interne Verwaltung ersichtlich ist. Der Dokumenten-Cache kann auch transient bzw. hauptspeicherbasiert verwendet werden und bietet für diesen Fall eine einfache DOM-basierte XPath-Implementierung an. Darüber hinaus sind aber noch eine Vielzahl weiterer Adaptoren denkbar bzw. sinnvoll, beispielsweise Adaptoren zur

- Validierung der registrierten Dokumente
- Transformation mittels XSLT
- Kompensation von Änderungsdiensten
- Kompensation fehlender Transaktionseigenschaften
- Unterstützung von Protokollen für entfernte Zugriffe (Corba, EJBs, Connector).

Interessant allerdings sind die Konzepte zur Konfiguration bzw. Spezifikation einer Abbildungsfunktion zur Kombination von, besonders nachträglich registrierter, Wrapper oder Adaptoren. Dieses wird im Folgenden weiter verfolgt.

3.6.3 SHARX Data Provider

In SHARX bilden also die Datenquellen bzw. Datenverwaltungssysteme zusammen mit einem jeweiligen spezifischen XML-Wrapper die so genannten Data-Provider-Komponente. Der XML-Wrapper stellt dabei dem SHARX-System eine XML:DB-Schnittstellenimplementierung bereit. Das Verhalten der Implementierung kann durch Adaptoren an die Erfordernisse der Anfrageverarbeitung angepasst werden (beispielsweise durch Cache-Unterstützung oder Kompensation) (siehe Abb. 3.10). Das Ziel bei der Anbindung der jeweiligen Datenquelle besteht darin, die Eigenschaften und Möglichkeiten der zugehörigen Datenverwaltungssysteme zu berücksichtigen und, wenn möglich, auch zu nutzen, um beispielsweise Teilanfragen entsprechend der Eigenschaften und Möglichkeiten der Datenquellen auch an diese delegieren zu können. Besitzt eine Datenquelle nicht die Fähigkeit, Anfragen zu beantworten, so kann diese Verarbeitung in der Mediator-Komponente durchgeführt werden oder in einem diese Funktionalität kompensierendem Adaptor. Die Entscheidung obliegt der Komponente XML-Integrator und kann zur Laufzeit während der eigentlichen Integration getroffen werden. Sie ist verantwortlich für die Ausführung und Auswahl der Integrationsstrategie unter Berücksichtigung der Eigenschaften und Möglichkeiten der jeweiligen Datenquellen. Die Unterstützung durch eine flexible Kopplung mittels XML Data Provider wurde in [Sie01] und [Höh01] vorgestellt. Die Beschreibung der Möglichkeiten und die Strategie zum Finden einer entsprechenden Konfiguration von XML-Wrapper und Adaptoren in SHARX wird im Folgenden erläutert.

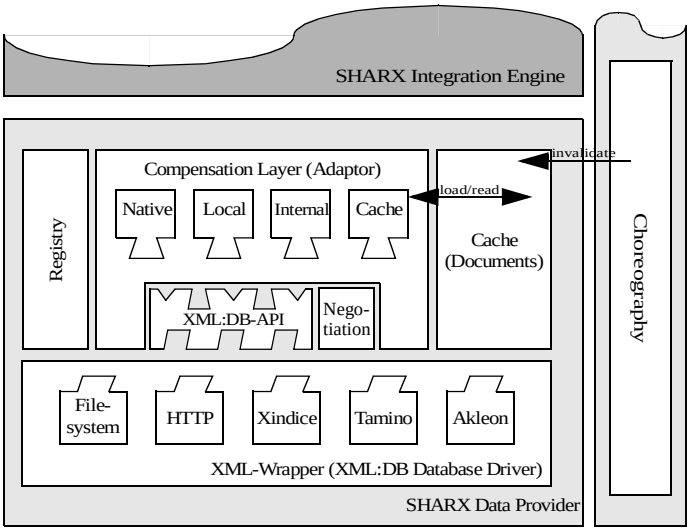


Abb. 3.10: SHARX Data Provider

3.6.4 Datenquellenbeschreibung

Bei der Anbindung der jeweiligen Datenquelle sollen die Fähigkeiten und Eigenschaften des zugehörigen Datenverwaltungssystems berücksichtigt werden. Ein XML-Data Provider registriert die von ihm verwalteten Dokumente bei der Registrierungskomponente zusammen mit den entsprechenden Metadaten, also den besonderen Eigenschaften eines Dokuments bzw. des Datenverwaltungssystems. Aber auch XML-Wrapper („Data Provider Class“) und Adaptoren sind im Repositorium zusammen mit ihren beschreibenden Attributen (Metadaten) registriert. Bei den Metadaten kann man zwischen den Fähigkeiten („Capabilities“) und Eigenschaften („Properties“) unterschei-

den. Sowohl Datenquellen als auch XML-Wrapper und zugehörige Adaptoren werden durch Fähigkeiten und Eigenschaften beschrieben, deren Anzahl flexibel erweitert werden kann. Die in den vorangegangenen Abschnitten genannten Komponenten kennen bzw. definieren die folgende Fähigkeiten und Eigenschaften.

Fähigkeiten

Fähigkeiten beschreiben die realisierte Funktionalität der XML:DB-Schnittstelle, die durch einen XML-Wrapper genutzt werden kann bzw. von einem Datenverwaltungssystems überhaupt unterstützt wird:

- XML-API: SAX, DOM oder TEXT
- XPath-Query
- XUpdate
- Collection Service Support
- Transaction Service Support.

Sie geben an, welche Methoden der XML:DB-Schnittstelle sinnvoll verwendet werden können und welche Dienste nutzbar sind. In SHARX wird eine Datenquelle nur sehr grobgranular beschrieben. Es ist auch denkbar, beispielsweise Anfragemöglichkeiten näher spezifizieren zu können, so dass auch ausgedrückt werden könnte, ob sich eine Anfrage nur auf bestimmte Attribute oder Attributgruppen beziehen darf. Dies ist bei der Integration von Web-Formularen sehr nützlich. Allerdings werden in SHARX diese Punkte als Wrapper-interne Aspekte behandelt und bei der Anfrageverarbeitung nicht berücksichtigt, d. h., sollten nicht alle denkbaren Prädikate oder Kombinationen möglich sein, dann muss der Wrapper die Anfrage weiter umschreiben und zunächst die Dokumente oder Teile davon holen und die Anfrage lokal vervollständigen.

Eigenschaften

Während sich Fähigkeiten auf die Beschreibung der Funktionalität bezüglich der XML:DB-Schnittstelle beschränken, beschreiben Eigenschaften darüber hinaus das Verhalten der Datenquellen in Bezug auf die zu verwaltenden Dokumente. Eigenschaftenbeschreibende Attribute werden somit auf Dokumentebene spezifiziert. Folgende Eigenschaften werden von den bisherigen XML-Wrapper- und Adaptor-Implementierungen in SHARX berücksichtigt:

- STATIC
- READONLY
- VIRTUAL
- NOTVARYING
- CACHELEVEL.

Dabei bedeutet STATIC, dass ein Dokument für den gesamten Zeitraum, für das es registriert ist, sich nicht ändert, d. h., auch Server-seitig findet keine Änderung statt. READONLY hingegen macht eine Aussage darüber, ob ein Dokument geändert werden kann und zeigt an, dass über deskriptive XUpdate-Anweisungen oder die XML-Resource-spezifischen Methoden der XML:DB-Schnittstelle Daten aktualisiert werden können. In diesem Fall tritt SHARX als Client gegenüber dem Datenverwaltungssystem auf. Natürlich können auch bei READONLY Server-seitig Änderun-

gen auftreten, d. h., dass ein Dokument nicht unbedingt statisch sein muss, wenn spezifiziert wurde nur lesend darauf zuzugreifen. Daher schliessen sich auch READONLY-Eigenschaft und XUpdate-Fähigkeit nicht von vornherein aus, denn über eine XUpdate-Operation kann das Dokument Server-seitig durch eine Client-Anforderung geändert werden. READONLY kann bei Einsatz von Nur-Lese-Speichern oder aber auch Sichten (s. a. VIRTUAL) verwendet werden. Ist das zugrunde liegende Dokument eine Sicht oder ein virtuelles Dokument, das nicht über die XML-Resource-spezifischen Methoden der XML:DB-Schnittstelle geändert werden kann, wohl aber über deskriptive XUpdate-Anweisungen, dann wird es mit dem Attribute VIRTUAL belegt. VIRTUAL zeigt an, dass es sich bei dem Dokument um eine Sicht handelt auf die nur mittels Anfragen zugegriffen werden kann und dass das Dokument nicht vollständig materialisierbar ist. Mit NOTVARYING kann in diesem Fall angezeigt werden, dass bei gleichen Anfragen mittels XPATH immer das gleiche Ergebnis zurückgeliefert wird. Dadurch wird ein eventuelles Result Caching in der Query Engine unterstützt. Caching auf Dokument- bzw. Data-Provider-Ebene wird durch den Caching-Adaptor unterstützt. Um datenquellenseitig Einfluß auf die Steuerung des Cache-Verhaltens zu nehmen und Hinweise für eine Optimierung zu liefern, ist im Zuge der Registrierung CACHELEVEL zu verwenden. Diese Eigenschaft zeigt an, welche Cache-Unterstützung durch SHARX sinnvoll erscheint. Dieses Attribut kann mehrere Werte annehmen, während die bisherigen Attribute nur boolsche Werte annehmen können. NOCACHE bedeutet, dass das Dokument nicht lokal im Cache gehalten werden darf, bei ONDEMAND liegt die Entscheidung bei SHARX und bei PRELOAD sollte sich das System zur Laufzeit genauso verhalten wie bei ONDEMAND, außer dass dadurch angezeigt wird, dass das Dokument bei der Registrierung in den Cache zu laden ist. PRELOAD ist als Verstärkung bzw. Spezialisierung von ONDEMAND zu verstehen, so dass beispielsweise bei mit STATIC deklarierten Dokumenten diese lokal im Cache verwaltet werden können. PRELOAD gibt einen Hinweis darauf, dass das Dokument bei Registrierung bereitsteht und bereits angefordert werden kann. Dies kann z. B. bei dynamisch erzeugten Dokumenten sinnvoll sein, die erst aufwändig generiert werden müssen, aber während der weiteren Verarbeitung in SHARX statisch sind. Die Verstärkung von PRELOAD ist MATERIALIZE. In diesem Fall kann und soll das Dokument lokal gespeichert werden. Aber zusätzlich zu der lokalen Speicherung in einem Dokumenten-Cache kommt bei MATERIALIZE hinzu, dass die Ressource nicht im Cache vorgehalten wird, sondern persistent in einem internen XML-Datenverwaltungssystem verwaltet wird.

Anpassung (Negotiation)

Das Auffinden einer sinnvollen Konfiguration von Datenquellen, XML-Wrapper und Adaptoren folgt dabei einem einfachen Regelwerk, das aus drei verschiedenartigen Regeln besteht: Abbildungsregeln (Mapping-Rules), Anpassungsregeln (Matching-Rules) und einer speziellen Abbildungsregel für einen Vorgabewert (Default-Rule). Sollte es mehr als einen Adaptor geben, der sich qualifiziert, werden zuerst die Abbildungsregeln ausgewertet. Qualifiziert sich genau ein Adaptor ist eine Konfiguration gefunden. Kann nicht eindeutig entschieden werden, welcher Adaptor sich qualifiziert, wird anhand der Anpassungsregeln derjenige ausgesucht, der die Anforderungen bestmöglich erfüllt. Dies wird anhand einer Ähnlichkeits- bzw. Überdeckungsfunktion („Coverage“) bestimmt. Die Regel mit der höchsten Überdeckung wird angewendet und der entsprechende Adaptor ausgewählt. Wenn der Wert nicht signifikant ist, also nicht hoch genug ist und einen Schwell-

wert nicht überschreitet, wird die Regel nicht angewendet. Dieser Schwellwert ist Teil der Anpassungsregel. Qualifiziert sich wieder kein Adaptor, so wird in einem letzten Schritt überprüft, ob der dem Vorgabewert entsprechende Adaptor (Default-Rule) in der Menge der möglichen Adaptoren liegt. Ist dies der Fall, wird diese Regel angewendet. Qualifiziert sich kein Adaptor, so wird die Registrierung zurückgewiesen. Diese Auswertung kann auch im laufenden Betrieb noch angepasst werden, z. B. wenn Änderungen auf einem Dokument durchgeführt werden sollen, das nicht änderbar ist. Es kann überprüft werden, ob ein passender, die Integritätsbedingungen erfüllender, kompensierender Adaptor existiert, der diese Funktionalität bereitstellt.

3.7 Registry-Komponente

Bei der Registrierung einer Datenquelle kann eine Vielzahl von Informationen angegeben werden. Es sollte festgelegt werden, wie die Datenquelle angesprochen werden kann; dazu ist eine URL und ein XML-Wrapper zu spezifizieren. Ferner sind die Fähigkeiten der Datenquelle, die bereitgestellten Sammlungen („Collection“) und die enthaltenen Dokumente mit ihren jeweiligen Namen bzw. Surrogaten („ID“) anzugeben. Dies geschieht mittels spezifischer Kommunikationsprotokolle (RMI/IIOP) oder über eine Web-Service-Schnittstelle in der Registry-Komponente. Für jede registrierte Datenquelle ist nachfolgend ein entsprechender Adaptor auszuwählen.

3.8 Choreography-Komponente

Das Ereignis der Registrierung einer Datenquelle muss an andere Komponenten weitergeleitet werden. So ist beispielsweise der Integration Engine oder die Caching-Komponente über diese und andere unten genannten Änderungen zu informieren, um die Schemabeschreibungen, die Partitionierung und die Integration anzupassen. In SHARX können verschiedende Ereignisse in unterschiedlicher Ausprägung auftreten:

- Veränderungen in den Daten ohne Strukturveränderung (Ändern),
- Veränderungen in der Struktur (Einfügen, Ändern, Löschen) oder
- Veränderungen in den Metadaten (Einfügen, Ändern, Löschen).

Die Choreography-Komponente wird über diese Ereignisse unterrichtet und propagiert diese Nachrichten an die entsprechenden Komponenten, d. h., sie ist damit auch für die nachfolgenden Aktionen verantwortlich und koordiniert die weitere Verarbeitung. In anderen System (z. B. Enosys) spricht man daher auch von Notification Back-End. Die Registrierung und entsprechende Adaptoren (Cache) sind die einzigen Nachrichtenquellen der Choreography-Komponente. Nur sie lösen systeminterne Prozesse aus, die durch die Choreography-Komponente gesteuert werden. Nach einer Registrierung ist zunächst die Integration Engine zu benachrichtigen.

3.9 SHARX Integration Engine

Die dem Partition Miner zugrunde liegenden Konzepte stellen eine Kombination aus Partitionierung- (oder auch Clustering-) und Data-Mining-Konzepten dar. Die Analyse der Menge der Dokumente durch den Partition Miner hat zum Ziel, disjunkte Partitionen ähnlicher Dokumente in der Gesamtmenge aller zu integrierender Dokumente zu finden. Bei der Analyse werden Informationen

über die jeweiligen Inhaltsmodelle basierend auf DTDs oder XML-Schemata [XSD00], eingesetzte Namensräume und verwendete Elementnamen benötigt. Eine effiziente Analyse dieser Informationen erfordert eine geeignete Unterstützung. Die Repository-Komponente realisiert diese Unterstützung, indem sie Informationen zu allen verfügbaren Dokumenten verwaltet. Es werden zusammen mit den entsprechenden Metadaten, wie z. B. Dokumenttyp, verwendete Elementnamen oder Größe, auch Informationen über den Speicherungsort, also Name der Datenquelle und ihre URI, verwaltet. Ähnlichkeit definiert sich in diesem Zusammenhang allerdings nicht über die Semantik von Elementen, sondern über die zugrunde liegende Struktur. Die Semantik der Elemente und damit die Berücksichtigung beispielsweise von Synonymen geschieht durch die Integration von Beziehungen zum Zeitpunkt der Anfrageverarbeitung durch sogenannte navigationsbezogene Dokumente und zählt damit in SHARX zu den dynamischen Integrationsaspekten.

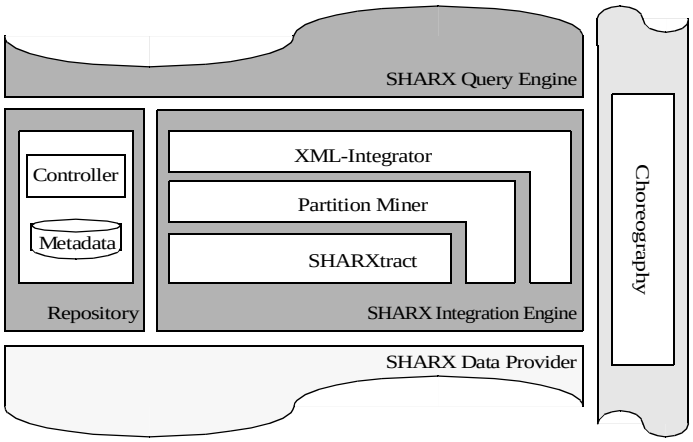


Abb. 3.11: SHARX Integration Engine

Insgesamt erlaubt dies durch die spätere Einflussnahme durch den Benutzer auf den Prozess der Integration ein umfassendes Personalisierungskonzept. Die Integration unter Berücksichtigung der statischen Aspekte entspricht weitestgehend einer Schemaintegration, allerdings unter Berücksichtigung von in XML-Dokumenten vorkommenden Referenzen (ID/IDREF(S), KEY/KEYREF oder auch XLink). Von diesen Referenzen werden in SHARX die Beziehungen abgeleitet. Die drei Schritte der Schemaextraktion, der Partitionierung und der Integration werden durch die Komponenten der Integration Engine realisiert. Registriert eine Datenquelle zu verarbeitende XML-Dokumente, wird zunächst der Schemaextraktor (SHARXtract) durch die Choreography-Komponente darüber informiert und eine interne Strukturrepräsentation erstellt und im Repository abgelegt. Diese dient der Partitionierung (Partition Miner) und der Integration als Grundlage der weiteren Verarbeitung.

3.9.1 Schemaextraktor: SHARXtract

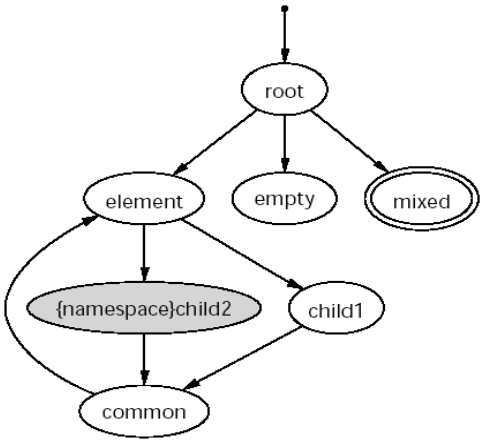
Die Voraussetzung des Partition Mining besteht aus der Verfügbarkeit von Schemainformationen. Es soll aber auch die Klasse von Dokumenten berücksichtigt werden, die kein explizites Schema besitzen. In diesem Fall werden Schemainformationen aus den Dokumenten selbst gewonnen und mit den Informationen,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

- Klassifikation der Dokumente in die SHARX-spezifischen Nutzungsklassen
- Berücksichtigung von Namespaces
- sinnvolle Behandlung von Mixed Content bei datenorientierter Sicht auf XML-Dokumente
- Extraktion der Schemainformationen aus XML-Dokumenten selbst
- Ausnutzung vorhandener Strukturinformationen (DTDs, XML-Schema-Dokumente)
- optimierte Speicherung der Schemainformationen.

Durch die Schemaextraktion werden Partition Miner und XML-Integrator 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 XML-Integrator relevanten Strukturen reduzieren. In diesem Sinne erfüllt der Dienst Schemaextraktion eine komplexitätsreduzierende und vorverarbeitende, aber auch eine strukturerschließende Funktion, insbesondere durch Namespace-Informationen [Bue02].

Schemarepräsentation

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 so genannten DataGuides, die zur strukturellen Zusammenfassung von semistrukturierten Datenbanken entwickelt wurden [GW97]. Abb. 3.12 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 >
      <ns:child2 xmlns:ns="namespace" />
   </element>
   <empty/>
   <mixed>
      Dies ist <em>Mixed</em> Content.
   </mixed>
</root>
```

Abb. 3.12: Schemagraph zu einem einfachen XML-Dokument

Das Modell zur Repräsentation von Schemainformationen kennt vier besondere Knotentypen, wobei jeder Knoten eine Kombination aus einem Namen und einer Namespace-Definition besitzt, die einen Knoten eindeutig identifizieren, aber auch leer sein darf. Ein Elementknoten bildet die Wurzel. Dieser Wurzelknoten, im Beispiel mit „root“ benannt, hat genau eine eingehende Kante und kann weitere Elementknoten referenzieren. Die referenzierten Elementknoten können entsprechend ihrem Vorkommen im Dokument eine einfache (weiß, ohne Rand) oder eine mehrfache Kardinalität (grau) besitzen. Mixed-Content-Elementknoten werden durch einen zusätzlichen Rand gekennzeichnet. Während ein Wurzelknoten auch Mixed-Content beinhalten kann, so verbietet allerdings die XML-Spezifikation mehrere Wurzelelemente, so dass der Wurzelknoten immer die Kardinalität eins besitzt. Der zugehörige Namespace eines Elements ist in geschweiften Klammern dem Namen vorangesetzt. Die Reihenfolge der Elemente ist in SHARX nicht von Bedeutung. Allerdings kann dies insbesondere bei Mixed Content zu einem wenig sinnvollen Integrationsverhalten führen. Um Mixed Content bei der Partitionierung und Integration entsprechend behandeln zu können, werden diese Knoten markiert. Es wird aber dennoch das gesamte Schema extrahiert bzw. transformiert,

um nicht nur diese statischen Integrationsaspekte abzudecken, sondern auch die Anfrageverarbeitung zu unterstützen, die zu den dynamischen Integrationsaspekten zählt.

3.9.2 Schematransformation und -Extraktion

Der in Abb. 3.12 gezeigte Graph kann auf unterschiedliche Art und Weise abgeleitet werden. Die Daten können direkt aus dem XML-Dokument gewonnen werden, wenn z. B. keine weiteren Schemainformationen verfügbar sind. Dies ist allerdings immer mit dem Holen und einer vollständigen Analyse (Parsing) des gesamten Dokuments verbunden. Die Analyse ist hingegen nicht sehr anspruchsvoll. Ausgehend vom Wurzelelement wird das gesamte Dokument auf das Vorkommen von Elementen und Kindelementen untersucht. Das Auftreten eines Elements in der internen Schemarepräsentation wird vermerkt und entsprechende Knoten in den korrespondierenden Graphen eingefügt, die gemäß der bestehenden Beziehungsverhältnisse durch Kanten verbunden werden. Ein Knoten ist durch den Namensraum und den lokalen Namen eines Elements eindeutig bestimmt, d. h., jede Kombination aus Namensraum und Elementnamen existiert nur genau einmal (siehe Abb. 3.12). Ist aber eine DTD im Dokument angegeben, so kann eventuell auf die Analyse des vollständigen Dokuments verzichtet werden, wenn die benötigten Namespace-Definitionen vollständig in der DTD enthalten sind. Werden Namespace-Definitionen durch Angaben im Dokument ergänzt, muss das Dokument in die Analyse einbezogen werden. Da sich eine Namespace-Definition nicht allein auf ein Element, sondern auch auf alle darunter liegenden Kindelemente auswirken kann, ist es in manchen Fällen sinnvoll, ein Beispieldokument zu generieren, das alle möglichen Belegungen abdeckt. Diese drei Fälle und die Behandlung von XML-Schema-Dokumenten wird im Folgenden weiter untersucht [Bue02].

DTD

Liegen explizite Strukturinformationen in Form einer DTD vor und sind alle Namespace-Definitionen vollständig als Konstanten in der DTD angegeben, so muss die vorliegende Strukturinformation nur in die interne Strukturbeschreibung transformiert werden. Das XML-Dokument muss nicht in die Analyse einbezogen werden. Für einige triviale Fälle lassen sich sehr einfach Transformationsvorschriften ableiten (siehe Beispiel in Abb. 3.13):

- Der Spezialfall des leeren Elements, dass also ein Element kein Kindelemente besitzt, erzeugt einen Knoten im Graph, der ebenfalls keine weiteren Knoten referenziert. Zu erkennen ist dieser Fall an der Verwendung des Schlüsselwort EMPTY in der Elementdefinition.
- Mixed Content in einer Elementdefinition zu erkennen, ist auch recht einfach möglich, da eine solche Definition mit dem Schlüsselwort #PCDATA im Content Model beginnt.
- Auch die Verwendung des Schlüsselwortes ANY in der Elementdefinition lässt einen Knoten der Interndarstellung zu einem Mixed-Content-Knoten werden. ANY zeigt an, dass beliebige Kindelemente im Content Model vorkommen können. Allerdings müssen es Elemente sein, die in der DTD auch definiert wurden. Dies schließt ebenso die Kombination von Text und Elementen mit ein. Daher muss angenommen werden, dass zumindest potenziell Mixed Content vorkommen kann. Im Schemagraph führt dann jeweils eine Kante von dem Knoten, der mit ANY belegt ist, zu allen anderen Knoten des Graphen (In Abb. 3.13 wurde der Übersichtlichkeit darauf aber verzichtet).

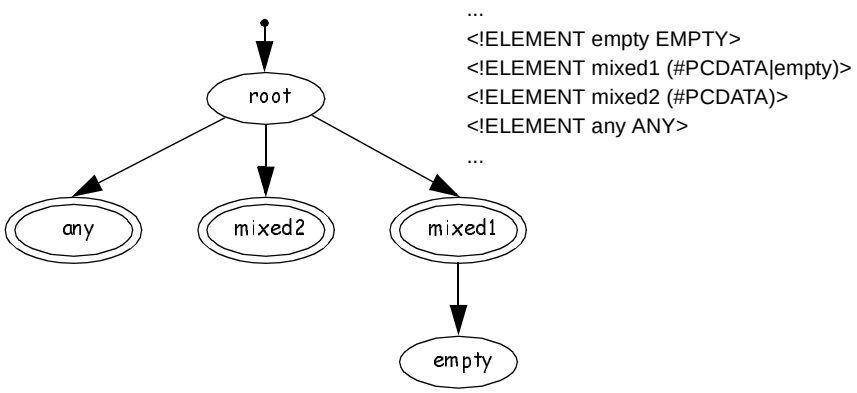


Abb. 3.13: Schemagraph zu einer einfachen DTD

Neben diesen einfachen und sehr speziellen Fällen, die ferner noch keine Namespace-Definitionen beinhalten, ist der allgemeine Fall des Element Content sehr viel prägnanter. Also der Fall, dass die möglichen Kindelemente, ihre Reihenfolge und Kardinalitäten für ein entsprechendes Vaterelement angegeben ist. Da die Reihenfolge und die Kardinalität in SHARX eine untergeordnete Rolle bei der Integration spielen, genügt es allein die Menge der Kinder mit den entsprechenden Namespace-Präfixen und den zugehörigen Namespace-Definitionen zu betrachten. Zu diesem Zweck reicht es aber nicht aus, wie in [Bue02] vorgeschlagen, aus dem Inhaltsmodell der DTD-Definition eines Elements nur die Menge aller vorkommenden Kindelemente zu bestimmen, z. B. reicht bei

(a, (b?, (c+ | d))*)

die Bestimmung von {a, b, c, d} nicht aus, um zu erkennen, dass Element a genau einmal vorkommen kann, alle anderen hingegen mehrfach. Die genaue Kardinalität spielt in SHARX zwar keine Rolle, allerdings muss zwischen mehrfachem und einfachem Auftreten eines Kindelements unterschieden werden. Ein einfaches Regelsystem leistet die erforderlichen Umformungen (Abb. 3.14).

Reduktion	
(A | B) => ((A)?, (B)?)	(GL 1)
(A, B)* => ((A)*, (B)*)	(GL 2)
((A)) => (A)	(GL 3)
(A, (B)) => (A, B)	(GL 4)
((A), B) => (A, (B))	(GL 5)
Vereinfachung	
A+ => A*	(GL 6)
A? => A	(GL 7)
(a)* => (a*)	(GL 8)
(a*)* => (a*)	(GL 9)
Gruppierung	
(..., a, ... , a, ...) => (a*, ...)	(GL 10)
(..., a*, ... , a, ...) => (a*, ...)	(GL 11)
(..., a, ... , a*, ...) => (a*, ...)	(GL 12)
(..., a*, ... , a*, ...) => (a*, ...)	(GL 13)

Abb. 3.14: Regelsystem zur Vereinfachung von DTDs

Dabei ist a ein Element aus der Menge aller in einem Inhaltsmodell vorkommen Kindelemente und A sowie B Terme, die über der Menge aller Elemente entsprechend der Produktionsregeln für DTDs gebildet werden können. In Anlehung an [STH+99] liegt auch hier die Idee zugrunde, die jeweiligen Inhaltsmodelle schrittweise so zu vereinfachen, dass ein Dokument, das gegen die Ursprungsform validiert werden konnte, auch gegen das Umformungsergebnis noch validiert werden kann. Dieses Prinzip trifft allerdings nur auf die Reduktion und Vereinfachung zu. Bei der Gruppierung und (GL 2) hingegen stimmt diese Aussage nur, wenn die Dokumentreihenfolge vernachlässigt wird. Das Inhaltsmodell verliert dabei zwar an Genauigkeit, allerdings besitzt das Umformungsergebnis letztendlich auch nicht mehr Ausdrucksmächtigkeit als die interne Schemarepräsentation und kann einfach in die Interndarstellung überführt werden. Ausgehend von obigem Beispiel ergibt sich also:

(a, (b?, (c+ | d))*)	(6)
=>(a, (b?, (c* | d))*)	(1)
=>(a, (b?, ((c*)?, (d)?))*)	(7)
=>(a, (b, ((c*), (d)))*)	(4)
=>(a, (b, ((c*), d))*)	(5)
=>(a, (b, (c*, (d)))*)	(4)
=>(a, (b, (c*, d))*)	(2)
=>(a, ((b)*, ((c*, d))*))	(3)
=>(a, ((b)*, (c*, d)*))	(8)
=>(a, ((b*), (c*, d)*))	(5)
=>(a, (b*, ((c*, d)*)))	(4)
=>(a, (b*, (c*, d)*))	(2)
=>(a, (b*, ((c*)*, (d)*)))	(9, 8)
=>(a, (b*, ((c*), (d*))))	(4)
=>(a, (b*, ((c*), d*)))	(5)
=>(a, (b*, (c*, (d*))))	(4)
=>(a, (b*, (c*, d*)))	(4)
=>(a, (b*, c*, d*))	(4)
=>(a, b*, c*, d*)	

Es ist nun einfach[12] zu erkennen, dass das erste Element (a) nur einmal vorkommen kann, aber alle anderen (b bis d) mehrfach. Dies wird nun unter Vernachlässigung der Reihenfolge auf den Graph übertragen. Die Berücksichtigung von Namespace-Definitionen und -Präfixen macht die Schematransformation noch um einiges interessanter (siehe [Bue02]).

12.) Auf die formal korrekte Eliminierung aller Klammern kann in einer Realisierung verzichtet werden, so dass sich die Menge der Regeln stark vereinfachen.

3.9.3 Partition Miner

Aufgabe des Partition Miner ist die Analyse der Menge der zu integrierenden Dokumente. Auf der Basis des internen Schemas wird die Menge der operationalen Dokumente auf Ähnlichkeiten untersucht und ähnlich strukturierte Dokumente zu Partitionen zusammengefasst. Partitionen unterteilen die Gesamtmenge aller Dokumente in disjunkte Teilmengen.

Dokument 1:
```
<publications xmlns="www.uni-kl.de/publications">
    <articles>
        <article>
            <title>Use Cases im SE</title>
            <author>Marcus Ciolkowski</author>
        </article>
    </articles>
    <books xmlns="www.uni-kl.de/books">
        <book>
            <title>Software Change Impact Analysis</title>
            <author>Antje von Knethen</author
        </book>
    </books>
</publications>
```
Dokument 2:
```
<article xmlns="www.uni-kl.de/publications">
    <title>Storing Semistructured Data with STORED</title>
</article>
```
Dokument 3:
```
<author xmlns="www.uni-kl.de/publications">John Doe</author>
```
Dokument 4:
```
<books xmlns="www.uni-kl.de/books">
    <book>
        <title>Use Cases</title>
    </book>
</books>
```

Abb. 3.14: Beispieldokumente (Teil 1)

Strukturierende und navigationsbezogene Dokumente bilden eine eigene Partition. Partitionen bilden die Grundlage der Erzeugung des Unified View. Die Konzepte der Partitionierung sollen zunächst an einem Beispiel motiviert werden, das sich einem Beschreibungsmodell bedient, welches die Grundidee des Unified View auf Partitionen überträgt. Kann man sich den Unified View als ein einziges XML-Dokument vorstellen, so kann auch eine Partition durch ein XML-Dokument repräsentiert werden. Allerdings wird keine materialisierte Sicht auf den Inhalt bzw. die Daten der jeweiligen Dokumente erzeugt, sondern eine Beschreibung der Partitionen gegeben.

Diese Partitionsbeschreibung gibt Auskunft über die enthaltenen Dokumente und wie mittels XInclude und einem optionalen XPath-Ausdruck auf sie zugegriffen werden kann, ohne sie notwendigerweise in dem beschreibenden Dokument materialisieren zu müssen. Die Partitionierung kann vollständig auf dem internen Schema arbeiten und erfordert nicht die erneute Analyse der XML-Dokumente. Aufgrund der Vorleistung der Schemaextraktionskomponente ist die Überprüfung auf eine vollständige oder teilweise Überdeckung recht einfach, da das interne Schema hierzu direkt Auskunft geben kann. Die Auswertung des internen Schemas soll im Folgenden durch ein Beispiel illustriert werden. Die Dokumente 1 bis 10, die in Abb. 3.14 und Abb. 3.15 dargestellt sind, werden zu diesem Zweck zunächst von der Schemaextraktionskomponente verarbeitet und nachfolgend anhand des abgeleiteten Schemas partitioniert.

Dokument 5:

```
<author id="12" xmlns="www.uni-kl.de/books">
    <fname>Tom</fname>
    <lname>Bohler</lname>
</author>
```

Dokument 6:

```
<publications xmlns="www.uni-kl.de/publications">
    <articles>
        <article>
            <title>Nutzung von Erweiterbarkeitsmechanismen</title>
            <author>
                <fname>Henrik</fname>
                <lname>Loeser</lname>
            </author>
            <author>
                <fname>Marcus</fname>
                <lname>Flehmig</lname>
            </author>
        </article>
    </articles>
</publications>
```

Dokument 7:

```
<authors xmlns="www.uni-kl.de/publications">
    <author>
        <fname>Marc</fname>
        <lname>Bohler</lname>
    </author>
</authors>
```

Dokument 8:

```
<root xmlns="www.uni-kl.de/books">
    <author>Marcus Flehmig</author>
    <person>Bruce Eckel</person>
</root>
```

Dokument 9:

```
<authors xmlns="www.uni-kl.de/books">
    <author>Wolfgang Mahnke</author>
</authors>
```

Dokument 10:

```
<people xmlns="www.uni-kl.de/books">
    <person>Bruce Wayne</person>
</people>
```

Abb. 3.15: Beispieldokumente (Teil 2)

Das interne Schema ist in Abb. 3.16 dargestellt. Mixed-Content-Knoten sind besonders gekennzeichnet. Die Elemente, die einem Mixed-Content-Knoten nachfolgen (<fname>, <lname>), sind für die Partitionierung nicht relevant, da sie nie ein Integrationspunkt sein können bzw. dürfen. In dem Beispiel ergibt sich durch Dokument 3 und 7, dass ein <author>-Element sowohl Text als auch Elemente zum Inhalt haben kann. Dieses Element muss somit als Mixed Content markiert werden und bei der weiteren Partitionierung ignoriert werden. Die einem Mixed-Content-Knoten nachfolgenden Knoten sind im Schemagraph in Abb. 3.16 daher nicht abgebildet. Allerdings wird die Schemaextraktion immer vollständig ausgeführt, denn die Informationen über nachfolgende Knoten sind für die Anfrageverarbeitung essentiell. Aus dem Graph des internen Schemas ist erkennbar, dass jedes Dokument mit mindestens einem anderen Dokument ein Element gemeinsam hat. Alle Dokumente stehen also in Beziehung und sind somit potentiell integrierbar. Daher wird die Menge der Dokumente nun in disjunkte Teilmengen zerlegt. Da die jeweiligen Schemata der Dokumente 2 und 4 vollständig von dem gemeinsamen Schema der Dokumente 1 und 6 überdeckt werden, beschreiben sie nur einen Teil dessen, was die Dokumente 1 und 6 beschreiben. Desweiteren kommt in den Pfaden von jeweils dem Wurzelelement des Schemas der Dokumente 2 und 4 bis zum Element <publications> jedes Element mit der Kardinalität 1 vor, also die Knoten <publications>, <books> und <articles>. Somit können die Dokumente 1, 2, 4 und 6 zur Partition 1 integriert werden. Da aber nun die Elemente <article> und <book> mehrfach vorkommen können, ist dies entsprechend im Schema zu vermerken. Das Dokument 10 ist nicht mit den Dokumenten der Partition 1 integrierbar, da kein gemeinsamer Knoten existiert. Die restlichen Dokumente können ebenfalls nicht der Partition 1 zugeordnet werden, da die Elemente <book> und <article> der Partition 1 mehrmals vorkommen können und somit ohne zusätzliche Informationen nicht festgestellt werden kann, welcher Autor dieser Dokumente welchem Buch zugeordnet werden soll. Dies erfordert die Multiplizität eines Knotens kennen zu müs-

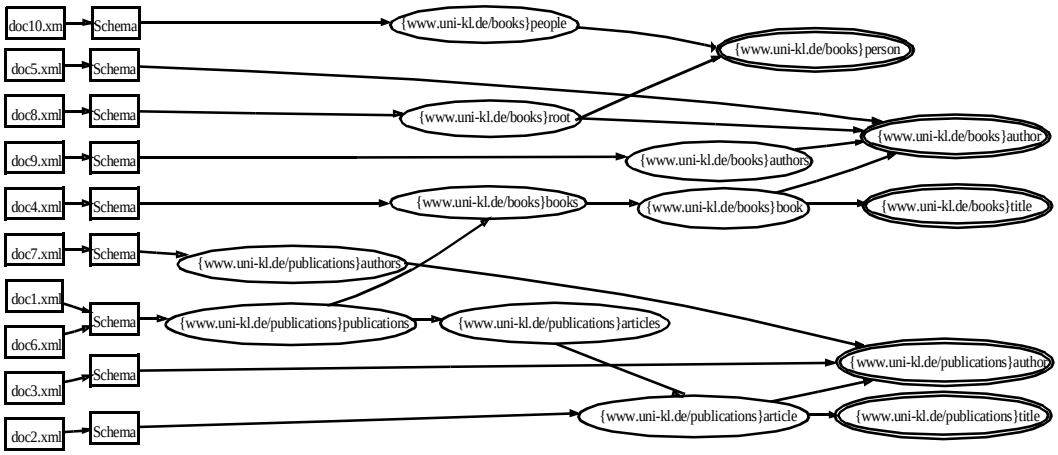


Abb. 3.16: Ergebnis der Schemaextraktion

sen, d. h., im internen Schema wird zwischen einfach und mehrfach auftretenden Elementen unterschieden.

```
<?xml version='1.0'?>
<partition:partition xmlns:xi="http://www.w3.org/2001/XInclude“
        xmlns:partition=”http://www.uni-kl.de/partitions”
        partition:id=“Partition1“ partition:documentclass=“operational“>
   <publications xmlns=“www.uni-kl.de/publications“>
        <articles>
            <xi:include href="doc1.xml#xpointer(/publications/articles/article)"/>
            <xi:include href="doc2.xml#xpointer(/article)"/>
            <xi:include href="doc6.xml#xpointer(/publications/articles/article)“/>
        </articles>
        <books xmlns=“www.uni-kl.de/publications“>
            <xi:include href="doc1.xml#xpointer(/publications/books/book)"/>
            <xi:include href="doc4.xml#xpointer(/books/book)"/>
        </books>
   </publications>
</partition:partition>
```

Abb. 3.17: Beschreibung der Partition 1

Nachdem nun die Partition 1 gebildet wurde (siehe Abb. 3.17), werden wieder Dokumente gesucht, die zu einer weiteren Partition integrierbar sind. Die Dokumente, die der Partition 1 angehören, müssen des Weiteren nicht mehr betrachtet werden. Die Dokumente 5, 8 und 9 beschreiben Autoren, die denselben Namensraum aufweisen, nur dass sie unterschiedliche Wurzelelemente besitzen. Die Dokumente sind offensichtlich vereinigungsfähig, da alle Dokumente Autoren beschreiben, nur die Wurzelknoten voneinander abweichen und diese aber die Kardinalität 1 besitzen. Diese Wurzelknoten könnten aber auf einen gemeinsamen Wurzelknoten, beispielsweise „authors“, reduziert werden, da in allen Dokumenten Informationen über Autoren zur Verfügung gestellt werden. Es ergeben sich aber die zwei folgenden Probleme:

1. Wie kann der Partition Miner entscheiden, welches Wurzelelement verwendet werden soll?
2. Wie kann der Partition Miner entscheiden, dass Dokumente ähnliche Daten beschreiben?

Da der Partition Miner nur lokales Wissen besitzt und über kein Wissen darüber verfügt, wie Partitionen integriert werden, kann das erste Problem nicht von dieser Komponente allein gelöst wer-

den. Der XML-Integrator verfügt allerdings über alle Informationen, die zur Lösung des Problems erforderlich sind. Würde der Partition Miner hingegen sich dieses Problems annehmen, indem er sich Wissen darüber verschaffen würde, wie die strukturierenden Dokumente aufgebaut sind und Partitionen integriert werden, so übernähme er Teil des Integrationsvorgangs vorweg. Desweiteren muss der XML-Integrator dieses Problem bei der Bestimmung der Einfügeposition einer Partition anhand der strukturierenden Dokumente ohnehin auch für andere Partitionen lösen. Es muss also zunächst nur eine Möglichkeit zur Beschreibung dieses Sachverhaltes gefunden werden. Zu diesem Zweck werden Partitionen von Partitionen, die wiederum aus so genannten Possible-Root-Partitionen bestehen, verwendet.

Um das zweite Problem zu lösen, sind die jeweiligen Schemata auf Überdeckung von Dokumenten hin zu untersuchen. Das Dokument 5 beschreibt einen Teil dessen, das auch Dokument 8 und 9 beschreiben, da sein Schema wieder einen Teil des Schemas der Dokumente 8 und 9 darstellt. Die Dokumente 3 und 7 allerdings können mit keinem der Dokumente 5, 8 und 9 integriert werden, da alle Knoten der beiden Schemata einem anderen Namenraum angehören und somit nach Definition als nicht gleich gelten. Die Schemata der Dokumente 8 und 9 weisen entlang des Pfades zum gemeinsamen Knoten „author“ alle die Kardinalität 1 auf. Daher kann der Knoten „author“ vom Dokument 5 sowohl mit dem Dokument 8 als auch mit dem Dokument 9 integriert werden. Es liegt somit ein Konflikt vor. Die Dokumente 8 und 9 weisen zwar einen gemeinsamen Knoten auf, aber das Schema des Dokumentes 8 wird nicht vollständig vom Schema des Dokuments 9 überdeckt. Für Dokument 9 gilt analog das Gleiche in Bezug auf Dokument 8. Beide können deshalb nicht zu einer Partition von Dokumenten kombiniert werden. Partition von Dokumenten enthalten ausschließlich Dokumente und können als Ganzes in den Unified View integriert werden. Aus diesem Grund werden die Dokument 5, 8 und 9 zu Partition 2 integriert, die allerdings einen anderen Aufbau als die Partition 1 aufweist. Sie enthält anstelle von Dokumenten, wie bereits angedeutet, Partitionen und wird daher eben als Partition von Partitionen bezeichnet. Eine einzelne Partition einer solchen Partition von Partitionen wird als Possible-Root-Partition bezeichnet. Eine „Partition von Partitionen“ wird nicht en bloc in den Unified View eingefügt, sondern zuerst in eine oder mehrere Partitionen von Dokumenten überführt. Dies ist aber nur mit weiterem Kontextwissen möglich, weshalb der XML-Integrator diese Aufgabe übernimmt.

Für alle Dokument, die sich teilweise überdecken, wird eine Possible-Root-Partition erstellt. Das Dokument 8 wird zur Partition R1 vom Typ „possibleRoot“ und das Dokument 9 zur Partition R2 vom Typ „possibleRoot“ integriert. Einer Possible-Root-Partition sind alle Dokumente zugeordnet, die mit den Dokumenten der Possible-Root-Partition integrierbar sind. Eine Possible-Root-Partition hat ebenso wie eine Partition von Dokumenten genau eine Wurzel, so dass nur die Dokumente zu einer Possible-Root-Partition integriert werden können, deren Schema eine Teilmenge des Schemas eines Dokuments der Possible-Root-Partition darstellt und die Kardinalität aller Knoten entlang des Pfades zum gemeinsamen Knoten 1 beträgt. Es kann vorkommen, dass mehrere Dokumente mehreren Possible-Root-Partitionen zugeordnet sind; in unserem Beispiel ist das Dokument 5 den Partitionen R1 und R2 zugeordnet, wie in Partition 2 im Abb. 3.18 zu erkennen ist.

Die Schemata des Dokuments 10 und 8 den gemeinsamen Knoten “{www.uni-kl.de/books}person”. Die Kardinalität alle Knoten bis zum gemeinsamen Knoten beträgt 1, weshalb die Dokumen-

```
<?xml version='1.0'?>
<partition:partition xmlns:xi="http://www.w3.org/2001/XInclude“
        xmlns:partition=”http://www.uni-kl.de/partitions”
        partition:id=“Partition2“ partition:documentclass=“operational“
        partition:partitionOfPartitions=“true“>

   <partition:possibleRoot partition:id=“R1“>
        <root xmlns=“www.uni-kl.de/books“>
            <xi:include href="doc8.xml#xpointer(/root/person)" parse="xml"/>
            <xi:include href="doc8.xml#xpointer(/root/author)" parse="xml"/>
            <xi:include href="doc5.xml#xpointer(/author)" parse="xml"/>
        </root>
   </partition:possibleRoot>
   <partition:possibleRoot partition:id=“R2“>
        <authors xmlns=“www.uni-kl.de/books“>
            <xi:include href="doc9.xml#xpointer(/authors/author)" parse="xml"/>
            <xi:include href="doc5.xml#xpointer(/author)" parse="xml"/>
        <authors>
   </partition:possibleRoot>
   <partition:possibleRoot partition:id=“R3“>
        <people xmlns=“www.uni-kl.de/books“>
            <xi:include href="doc10.xml#xpointer(/people/person)" parse="xml"/>
        </people>
   </partition:possibleRoot>
</partition:partition>
```

Abb. 3.18: Beschreibung der Partition 2

te integriert werden können. Das Schema von Dokument 10 stellt aber keine Teilmenge des Schemas von Dokument 8 dar, umgekehrt gilt dasselbe. Das Dokument 10 wird zur possibleRoot-Partition R3 integriert. Da das Dokument 8 bereits einer Partition von Partitionen zugeordnet ist, wird die Possible-Root-Partition R3 auch dieser Partition von Partitionen zugeordnet. Es handelt sich hierbei um die Partition 2. Nachdem nun die beiden ersten Partitionen erstellt wurden, werden die restlichen Dokumente auch untersucht. Die Dokumente 3 und 7 können zur Partition 3 zusammengefasst werden, da das Schema des Dokuments 3 eine Teilmenge des Schemas des Dokuments 7 darstellt. Desweiteren weist die Wurzel vom Dokument 7 die Kardinalität 1 auf (siehe Abb. 3.19).

```
<?xml version='1.0'?>
<partition:partition xmlns:xi="http://www.w3.org/2001/XInclude“
        xmlns:partition=”http://www.uni-kl.de/partitions”
        partition:id=“Partition3“ partition:documentclass=“operational“>

   <authors xmlns=“www.uni-kl.de/publications“>
        <xi:include href="doc3.xml#xpointer(/author)" parse="xml"/>
        <xi:include href="doc7.xml#xpointer(/authors/author)" parse="xml"/>
   </authors>
</partition:partition>
```

Abb. 3.19: Beschreibung der Partition 3

Partitionsbeschreibung

In den Abbildungen 3.17 bis 3.19 sind die zuvor beschriebenen Partitionen dargestellt. Sie können durch diese XML-Dokumente vollständig beschrieben werden. Beginnend mit einem <partition>-Wurzelelement finden sich zunächst gemeinsame, allein strukturtragende Elemente. Darin sind XInclude-Direktive eingebettet, die angeben, welche Teile der jeweiligen Dokumente an der entsprechenden Stelle integriert werden sollen. Die strukturtragenden Elemente sind insbesondere im Falle der Partition 2 und im Allgemeinen in den Fällen der Verwendung von Possible-Root-Partitionen wichtig. Ein Dokument kann in einer solchen Partition mehrfach referenziert sein, da es an verschiedenen Stellen der Partition eingefügt werden kann. Der notwendige Kontext, d. h., die Einfügeposition, wird durch die strukturtragenden Elemente definiert. Aber auch bei Partitionen von Dokumenten (Partition 1) kann es notwendig sein, ein Dokument mehrfach zu referenzieren. Existieren in einem Dokument Teilbäume, die mit anderen Dokumenten oder Teilbäumen integriert werden könnten, so muss an der entsprechenden Stelle der Partionsbeschreibung die jeweilige Stelle im Dokument angesprochen werden. Wie die XPath-Ausdrücke bestimmt werden können, folgt im nächsten Abschnitt. Dieses Verfahren wird auch bei der Bestimmung der oben verwendeten Partitionsbeschreibung angewendet und kann in [Boh03] nachgelesen werden. Allerdings ist es nicht besonders sinnvoll, Partitionsbeschreibung und Partitionsschema strikt zu trennen. Vielmehr dient hier die Partitionsbeschreibung als Beschreibungsmodell, d. h. als Illustration. Wesentlich sinnvoller ist eine Gleichbehandlung von Partitionsbeschreibung und Partitionsschema, da sie gleiche Vorgänge und Algorithmen verwenden und ähnliche Konzepte beschreiben. Eine mögliche Implementierung sollte ohnehin die Partitionsbeschreibung bzw. das Partitionsschema in einer leichter als XML zu verarbeitenden Form verwalten. SHARX verwendet hierzu ein relationales Datenbankverwaltungssystem und speichert Partitionsbeschreibung und Partitionsschema in einer gemeinsamen Struktur.

Definition

Zusammenfassend lässt sich somit sagen, dass bei der Partitionierung durch den Partition Miner die Menge der Dokumente in disjunkte Teilmengen unterteilt werden. Dabei wird ein Dokument genau einer Partition zugeordnet. Possible-Root-Partitionen entstehen, wenn die bei der Integration entstehende Struktur nicht eindeutig ist. Dabei können zwei Dokumente zu einer Partition zusammengefasst werden, wenn das Schema des ersten Dokumentes einen Knoten aufweist, der mit einem Knoten des Schemas des anderen Dokuments übereinstimmt (Überdeckung) und die Kardinalität aller Elemente von der Wurzel bis zu dem Knoten, der sich vor dem gemeinsamen Knoten befindet, 1 beträgt (Pfadeindeutigkeit). Die Kardinalität des gemeinsamen Knotens spielt keine Rolle. Die Gleichheit eines Elements definiert sich dabei über die Gleichheit von Namensraum und lokalem Namen. Die erstellte Partition ist eine Partition von Dokumenten, wenn der gemeinsame Knoten die Wurzel des Schemas einer der beiden Dokumente ist, andernfalls eine Partition von Partitionen. Ein Dokument kann mit einer Possible-Root-Partition oder einer Partition von Dokumenten integriert werden, wenn es zwischen der Wurzel des Schemas des Dokuments und einem beliebigen Knoten des Schemas der Partition zur Überdeckung kommt und das Kriterium der Pfadeindeutigkeit erfüllt ist. Dies führt unter Berücksichtigung des Auftretens von Mixed Content zu dem im Folgenden vorgestellten Algorithmus.

Partitionierungsalgorithmus

Aus der Betrachtung des Beispiels aus dem vorangegangenen Abschnitts soll im Folgenden ein Algorithmus abgeleitet werden. Bei dem Gesamtvorgang der Partitionierung ist grundsätzlich zuerst die Partition der strukturierenden Dokumente zu bilden. Anhand des vermerkten Dokumenttyps wird die Menge der entsprechenden Dokumente und daraus die Partition gebildet. Die Elemente an den Blättern in der Baumdarstellung des zugehörigen Schemas werden als mehrfach vorkommende Elemente im Schema vermerkt. Dies ist essentiell für die weitere Partitionierung, da diese Stellen Einfügepositionen bzw. Integrationspunkte darstellen und diese Elemente somit potenziell mehrfach vorkommen können. Des Weiteren müssen die operationalen Dokumente partitioniert werden. Zu diesem Zweck wird wieder die Menge aller zu partitionierenden Dokumenten gebildet, die nach der maximalen Tiefe des zugehörigen Schemas sortiert wird. Mit der Partitionierung wird mit dem Dokument mit der maximalen Tiefe begonnen. Dies liegt darin begründet, dass mit einem komplexeren Schema ein höherer Informationsgewinn zu erwarten ist und Anomalien bei der Integration durch dieses Vorgehen weitestgehend ausgeschlossen werden können. Es könnte bei der Partitionierung in dem vorangegangenen Beispiel mit Dokument 3 und 6 begonnen werden. Beide Dokumente haben gemeinsame Elemente (author), die bei einer Zuordnung zu einer gemeinsamen Partition die spätere Integration von Artikeln allerdings verhindern oder eine Neupartitionierung erfordern würde.

Die Partitionierung bekommt als Eingabe eine Strukturbeschreibung der zu verarbeitenden Dokumente und liefert als Ausgabe ein Menge von Partitionen. Dabei entstehen Partitionen von Dokumenten, wenn die in der Partitionen enthaltenen Dokumente strukturell voll verträglich sind. Für den Fall, dass kein eindeutiges Wurzelelement bestimmbar ist, entstehen mehrere Partitionen mit alternativen Wurzelelementen. Diese werden Possible-Root-Partitionen genannt und in einer Partition von Partitionen zusammengefasst. Um zu ermitteln, welche Dokumente strukturell ähnlich sind, bedient sich der Partition Miner folgender Kriterien: Zwei Dokumente sind genau dann kombinierbar, wenn

1. sie über gleiche Elemente verfügen (Überdeckung),
2. es von diesen Elementen (Integrationskandidaten) im jeweiligen Dokumentenschema einen Pfad zur ihrer Wurzel allein über Knoten der Kardinalität 1, die nicht vom Typ Mixed Content sind, gibt (Pfadeindeutigkeit) und
3. entweder die Integrationskandidaten selbst beide vom Typ Mixed Content sind oder keiner von beiden.

Da in SHARX Partitionsbeschreibung und Partitionsschema in einer gemeinsamen Struktur verwaltet werden, ist es naheliegend, nur einen Algorithmus zur Partitionierung zu betrachten. Eines der Hauptprobleme stellt die Suche nach Integrationskandidaten dar. Die Suche nach einem gemeinsamen Knoten mittels einer sequentiellen Suche über alle Knoten eines Partitionsschemas ist besonders für große Schemata sehr ungünstig. Es müssen sehr viele Vergleichsoperationen durchgeführt werden, um mögliche Kombinationsstellen zu ermitteln. Für den Fall, dass keine Übereinstimmung gefunden wird, hat dieses Verfahren die maximale Laufzeit, da alle Knoten abgesucht werden müssen. Unter der Voraussetzung aber, dass ein verwendetes Datenmodell für die Dokument- und Partitionsschemata eine schnelle und direkte Zugriffsmöglichkeit auf Knoten mit gleichem Namen und Namespace-Definition bietet, wäre es denkbar, diese Suche anders zu gestalten.

Es könnte zunächst eine Menge von Kandidaten ermitteln werden, die das Kriterium der Gleichheit erfüllen, über ein gleiches Inhaltsmodell verfügen sowie sowohl in der Partition als auch im Dokument vorkommen. Für jeden dieser Knoten müsste sowohl in der Partition als auch im Dokument die Pfadeindeutigkeit überprüft werden. Wird ein solcher Pfad gefunden, so handelt es sich nach Definition um einen Integrationskandidaten.

Die Leistung dieses Verfahrens hängt davon ab, wie viele gleiche Knoten in Dokument und Partition gefunden werden und überprüft werden müssen. Für den Fall, dass gar keine Überdeckung zwischen Dokument und Partition existiert und daher keine Integrationskandidaten gefunden werden können, ist das Verfahren sehr effizient. Für den Fall aber, dass viele mögliche Integrationskandidaten gefunden werden und für jeden dieser Knoten die Pfadeindeutigkeit geprüft werden muss, ist das Verfahren allerdings nicht effizienter und die Abarbeitung kann im schlechtesten Fall sogar die Laufzeit des oben geschilderten Verfahrens übersteigen. Dies ließe sich durch das Anlegen eines Index für alle Pfade, die das Kriterium der Pfadeindeutigkeit erfüllen, jedoch deutlich abmildern. Da allerdings davon auszugehen ist, dass die Menge der Kandidaten für die Integration eher überschaubar bleibt, ist zu erwarten, dass das zuletzt geschilderte Verfahren für typische Eingaben der Partitionierung effizienter ist.

Die bisherigen Überlegungen sind als Algorithmus in Tabelle 1 zusammengefasst. Dabei ist zu beachten, dass in Schritt 3a die Sonderbehandlung von Knoten des Inhaltstyps Mixed Content umgesetzt ist und in Schritt 4 die Entstehung und Behandlung von Partitionen von Partitionen durchgeführt wird.

3.9.4 XML-Integrator

In SHARX bestimmt die Partition der strukturierenden Dokumente die Grundstruktur des Unified View. Diese Struktur wird durch eine Menge von strukturierenden Dokumenten vorgegeben, die wie zuvor beschrieben partitioniert worden sind. Es sind besonders ausgezeichnete XML-Dokumente, die über den Namensraum des Wurzelelements gekennzeichnet sind. Sie bilden das Grundgerüst und damit den Ausgangspunkt des Unified View. Alle weiteren Partitionen werden an geeigneten Einfügepositionen in diesem Grundgerüst integriert. Dieser Vorgang findet im letzten Schritt der statischen Integration im XML-Integrator statt. Alle Daten der Partition müssen in der Unified View integriert werden (Vollständigkeitskriterium). Dabei gelten die Elemente und deren Nachfolger einer Partition als Daten, deren Kardinalität n ist. Allerdings dürfen diese Daten nur höchstens einmal in den Unified View integriert werden (Eindeutigkeit). Analog zu der Partitionierung können auch bei dem Vorgehen des XML-Integrators nur die Knoten des Schemas einer Partition als Einfügeposition dienen, die das Kriterium der Pfadeindeutigkeit erfüllen. Mit dem letzten Knoten eines geeigneten Pfades beginnen die Daten. Alle anderen Elemente, die „oberhalb“ einer Einfügeposition sich befinden, werden als strukturtragende Elemente bezeichnet. Bei den Partitionen kann es sich um Partitionen von Dokumenten oder aber Partitionen von Partitionen mit ihren zugehörigen Possible-Root-Partitionen handeln. Partitionen von Partitionen müssen gesondert behandelt werden, da mehrere Alternativen für die Wahl eines geeigneten Wurzelelements existieren und diese Mehrdeutigkeit zunächst aufgelöst werden muss. Bei der Suche nach einer geeigneten Einfügeposition im Grundgerüst, genügt es, nur die Blattknoten des Grundgerüstes zu betrachten.

Tabelle 1: Algorithmus zur Partitionierung einer Menge von Dokumenten

Schritt	Aufgabe
1	Initialisierung Erstelle eine Liste der noch zu bearbeitenden Dokumente einer gemeinsamen Klasse, die entsprechend der maximalen Schematiefe sortiert ist.
2	Erstellung der Partition Entferne das erste Dokument aus der Liste noch zu bearbeitender Dokumente und erstelle daraus eine neue Partition von Dokumenten. Dies geschieht durch die folgenden beiden Schritte: 1. Lege eine neue Beschreibung für die Partition an. 2. Kopiere das komplette Schema des Dokuments in die Beschreibung der Partition. Bei Erreichen eines Knoten mit der Kardinalität n: Füge einen Verweis auf das Dokument mit dem aktuellen Pfad hinzu. Die so neu erstellte Partition wird als Partition A bezeichnet.
3	Suche nach Integrationsknoten Untersuche nacheinander alle in der Liste noch zu bearbeitenden Dokumente auf Integrierbarkeit mit Partition A. Handelt es sich bei Partition A um eine Partition von Dokumenten, so wende Schritt 3a auf diese an; handelt es sich um eine Partition von Partitionen, so führe Schritt 3a für alle enthaltenen Possible-Root-Partitionen aus. Wird in 3a ein Integrationskandidaten gefunden, so führe Schritt 4 für den Integrationskandidaten aus. Ist dies nicht der Fall, so ist das Dokument nicht in die aktuelle Partition integrierbar. Fahre mit dem nächsten Dokument in Schritt 3 fort. Sind alle Dokumente in Schritt 3 auf Integrierbarkeit untersucht worden, so fahre fort mit Schritt 6.
3a	Suche nach gemeinsamen Knoten, die nach Definition gleich sind und entweder beide oder keiner von beiden als Mixed Content markiert sind. Füge dieser zu einer Liste von Integrationskandidaten hinzu. Für jeden dieser Kandidaten prüfe, ob es vom Kandidatenknoten einen Pfad zur Wurzel der Partition und des Dokuments gibt, für welchen alle enthaltenen Knoten die Kardinalität 1 aufweisen und nicht vom Typ Mixed Content sind. Wenn ja, dann vermerke diesen Pfad beim Kandidatenknoten. Kandidaten, für welche kein solcher Pfad existiert, werden aus der Menge entfernt. Ist die Kandidatenmenge leer, so wurde kein Integrationspunkt gefunden und eine Integration ist nicht möglich. Suche unter den verbliebenen Kandidaten den mit der geringsten Anzahl von Knoten im Pfad zur Wurzel der Partition, denn dieser ist der Integrationspunkt.
4	Integration Falls der Integrationsknoten die Wurzel des Dokuments ist, so fahre fort mit 4a. Falls der Integrationsknoten die Wurzel der Partition ist, so fahre fort mit 4b. Sonst fahre fort mit 4c.
4a	Erweitere das Schema der Partition unterhalb des Integrationsknoten um das Schema des Dokuments. Für alle Knoten im Dokument, deren Kardinalität n ist: Füge einen Verweis auf das Dokument mit dem aktuellen Pfad im Dokument hinzu.
4b	Erstelle aus dem Dokument eine neue Partition B nach den in Schritt 2 beschriebenen Anweisungen. Erweitere im Anschluss das Schema der Partition B am Integrationspunkt um das Schema der Partition A. Vorhandene Verweise auf Dokumente in Partition A werden übernommen und ihr Pfad wird angepasst.
4c	Falls Partition A eine Partition von Dokumenten ist, so erstelle eine Partition von Partitionen und füge Partition A zu dieser als Possible-Root-Partition hinzu. Von nun an wird diese Partition von Partitionen als Partition A bezeichnet. Erstelle aus dem Dokument wie in Schritt 2 beschrieben eine neue Partition und füge diese als Possible-Root-Partition zu Partition A hinzu.
5	Anpassung des Schemas Prüfe für alle eingefügten direkten Kindelemente unterhalb des Integrationspunkts, ob sie in der Partition mehrfach vorkommen können und markiere dies im Partitionsschema.
6	Eliminierung von Reihenfolgeabhängigkeiten Überprüfe erneut alle Dokumente aus der Liste noch zu bearbeitender Dokumente auf Integrierbarkeit wie in Schritt 3 beschrieben. Wird ein integrierbares Dokument gefunden, so führe die Integration wie in Schritt 4 geschildert durch, und führe Schritt 6 erneut aus. Wird kein integrierbares Dokument gefunden, so ist die Partitionierung für diese Dokumentklasse beendet.

In einem strukturierenden Dokument stellen daher alle Blätter der Baumstruktur des strukturierenden XML-Dokumentes eine mögliche Einfügeposition dar, alle anderen Knoten stellen strukturtragende Elemente dar. Die strukturtragenden Elemente weisen alle die Kardinalität 1 auf. Den Blättern hingegen wird immer die Kardinalität n zugewiesen.

Bei der Suche nach Einfügepositionen wird die Menge der Blattknoten des zu dem Grundgerüst gehörigen Schemas mit den Schemata der anderen Partitionen auf Überdeckung geprüft. Dabei werden Partitionen nicht immer en bloc eingefügt, sondern die Daten der Partitionen können in Form von Fragmenten an mehrere Stellen des Grundgerüstes in den Unified View eingefügt werden. Alle Fragmente einer Partition, die nicht an einer definierten Position eingefügt werden können, werden in einem Überlaufbereich des Unified View abgelegt. Um den Verwaltungsaufwand zu minimieren, sollten die Fragmente möglichst groß sein. Die Suche nach einem Knoten, der eine Einfügeposition besitzt, beginnt deshalb bei der Wurzel einer Partition und setzt sich rekursiv bei deren Kindelementen fort. Besitzt ein Knoten eine Einfügeposition, so spannt dieser ein Fragment auf, das den Knoten und rekursiv alle Kindknoten enthält. Diese Fragment kann dann an der gefundenen Einfügeposition eingefügt werden.

Algorithmus zur generische Datenintegration

Der XML-Integrator ist für das Einfügen der vorher in Partitionen eingeteilten Dokumente in eine gemeinsame Sicht verantwortlich. Zu diesem Zweck bekommt er als Eingabe eine Partition strukturierender Dokumente sowie Partitionen anderer Dokumenttypen und geht dann wie folgt vor:

1. Analyse der Partitionsbeschreibungen und Erstellen des Grundgerüstes des Unified View. Das Grundgerüst entspricht weitestgehend der Partition der strukturierenden Dokumente.
2. Suche für alle Partitionen anhand der Partitionsbeschreibung bzw. des Partitionsschemas eine Einfügeposition im Grundgerüst.
3. Erzeuge aus dem Grundgerüst und den Einfügepositionen eine Beschreibung des Unified View, die als interne Repräsentation der Unified View bezeichnet wird. Die interne Repräsentation stellt ein XML-Dokument dar, dessen Struktur der späteren Unified View entspricht und Referenzen auf Fragmente von Partitionen enthält. Das Auflösen dieser Referenzen führt zu einer Materialisierung des Unified View.

Im Einfügevorgang für Partitionen von Dokumenten wird versucht, rekursiv von der Wurzel der Partition aus möglichst große Teile der Partition einzufügen. Dabei wird die Möglichkeit der Kombination von Partition mit dem Unified View geprüft. Findet sich eine Überdeckung und ist das Kriterium der Pfadüberdeckung erfüllt, so stellt dies eine Einfügeposition dar. Die Suche nach Einfügepositionen beginnt mit der Wurzel der Partition und wird – insofern sich die Wurzel nicht einfügen lässt – in Form einer Breitensuche fortgesetzt. Wird eine Einfügeposition gefunden, so wird im entsprechenden Knoten der internen Repräsentation des Unified View ein Verweis auf die Partition mitsamt Pfadangabe zum integrierbaren Knoten eingefügt. Weiterhin wird auch im Knoten der Partition ein Verweis auf den Integrationspunkt in der internen Repräsentation des Unified View angelegt. Danach gilt der gesamte in der Partition darunter liegende Teilbaum als eingefügt. Mit der Suche nach einer Einfügeposition wird so lange fortgefahren, bis entweder alle Teilbäume eingefügt wurden, oder aber alle in Frage kommende Elemente der Partition untersucht wurden. Teilbäume, für welche keine Einfügeposition gefunden werden konnte, werden an einer speziellen Stel-

le im Grundgerüst des Unified View, dem sogenannten Überlauf-Bereich, untergebracht. Durch diese Maßnahme gehen Daten, welche sich nicht einfügen lassen, nicht für die gemeinsame Sicht verloren. Dieses Vorgehen ist in Tabelle 2 dargestellt.

Tabelle 2: Bestimmung der Einfügeposition für Partitionen von Dokumenten

Schritt	Aufgabe
1	Wende an allen Partitionen von Dokumenten den Schritt 2 für die Wurzel der Partition an. Die Wurzel stellt im Schritt 2 den Kontextknoten dar.
2	Hat der Kontextknoten der Partition eine Einfügeposition im Grundgerüst, so wende den Schritt 3 an und liefere dessen Ergebnis zurück. Ein Knoten des Schemas einer Partition besitzt eine Einfügeposition, wenn der Elementname und der Namensraum dieses Knotens mit dem eines Blattknotens des Grundgerüstes übereinstimmen und entweder beide vom Inhaltstyp Mixed Content sind oder beide nicht. Gibt es identische Blätter, so stellt der erste gefundene Blattknoten die Einfügeposition dar. Besitzt der Kontextknoten keine Einfügeposition, ist nicht vom Typ Mixed Content und stellt desweiteren die Kardinalität aller Knoten von der Wurzel der Partition bis zum Kontextknoten 1 dar, so wende den Schritt 2 auf allen Kindknoten des Kontextknotens an. Stellt das Ergebnis des Schrittes 2 für alle Knoten „OK“ dar, so liefere „OK“ zurück. Andernfalls vereinige alle Ergebnisse der Knoten, die nicht „OK“ zurücklieferten und liefere diese Vereinigung zurück. Besitzt ein Kontextknoten keine Kindknoten oder ist die Kardinalität des Kontextknotens n, so liefere einen XPath-Ausdruck zurück, der den Kontextknoten adressiert. Das Fragment, das durch den Kontextknoten aufgespannt wird, besitzt in diesem Fall keine Einfügeposition.
3	Im dritten Schritt wird einem Fragment seine Einfügeposition zugeordnet. Hierzu wird in der internen Repräsentation des Unified View ein Verweis auf die Partition und den Pfad des Kontextknotens in der Partition angebracht. Weiterhin wird das Schema der Partition unterhalb des Kontextknotens in die interne Repräsentation kopiert. Liefere das Ergebnis „OK“ zurück.

Die Bestimmung der Einfügeposition für eine Partition von Partitionen ist, wie eingangs bereits erwähnt, wesentlich aufwändiger, da Mehrdeutigkeit in Bezug auf das auszuwählende Wurzelelement zunächst aufgelöst muss. Die Grundidee dabei besteht darin, alle Possible-Root-Partitionen, für die sich alle enthaltenen Daten einfügen lassen, in Partitionen von Dokumenten umzuwandeln und aus der Partition von Partitionen zu entfernen. Für Possible-Root-Partitionen, die nicht komplett eingefügt werden können, muss ein Kompromiss eingegangen werden. Um alle Daten einzufügen, wird der Verlust von Strukturinformationen in Kauf genommen. Daher geht der XML-Integrator für die Partitionen folgendermassen vor: Könnten alle Daten der Partition eingefügt werden, wenn diese Teil einer anderen Partition wären, so fügt er die Daten zu dieser Partition hinzu. Ist dies nicht der Fall, so wird zwischen zwei Fällen unterschieden:

1. Überhaupt keine Daten der Possible-Root-Partition können ins Grundgerüst eingefügt werden, aber es können Teile in andere Partitionen eingefügt werden.
2. Nur ein Teil der Daten kann ins Grundgerüst eingefügt werden oder es können überhaupt keine Daten ins Grundgerüst und in andere Partitionen eingefügt werden.

Für den ersten Fall versucht der XML-Integrator so viele Daten wie möglich in andere Partitionen einzufügen. Die verbleibenden Daten werden dann zum Überlaufbereich hinzufügt. Im zweiten Fall wird die Possible-Root-Partition zu einer Partition von Dokumenten umgewandelt und es werden alle einfügbaren Daten in den Unified View eingefügt. Die verbliebenden Daten werden im Überlaufbereich des Unified View untergebracht. Tabelle 3 beschreibt dieses Vorgehen.

Das dreiphasige Vorgehen hat den Vorteil, dass die vollständig einfügbaren Possible-Root-Partitionen nach ihrer Umwandlung in Partitionen von Dokumenten nicht mehr modifiziert werden müssen, da diese Umwandlung erst in der letzten Phase geschieht. Ferner kann besser überprüft werden, ob das Einfügen aller Daten einer Partition A in eine Partition B garantieren kann, dass alle

Daten der ursprünglichen Partition in den Unified View gelangen. Hier würde als Partition B nur eine Possible-Root-Partition in Frage kommen, die zu Anfang in eine Partition von Dokumenten transformiert wurde.

Tabelle 3: Bestimmung der Einfügeposition für Partitionen von Partitionen

Schritt	Aufgabe
1	Wende an allen Partitionen von Partitionen den Schritt 2 an.
2	Suche für alle Possible-Root-Partitionen der Partition nach dem Algorithmus der Tabelle 2 Einfügepositionen im Grundgerüst und die dazugehörigen Fragmente. Teile die Partitionen in eine Menge von Partitionen, deren Daten sich vollständig einfügen lassen, und in eine Menge von Partitionen, deren Daten sich nicht vollständig einfügen lassen. Wende auf allen Possible-Root-Partitionen, die sich nicht vollständig einfügen lassen, den Schritt 4 an und wende im Anschluss auf allen Possible-Root-Partitionen, die sich vollständig einfügen lassen, den Schritt 3 an.
3	Einfügen von Possible-Root-Partitionen, für die alle Daten eingefügt werden können Entferne die Possible-Root-Partition aus der Partition von Partitionen und erzeuge aus ihr eine Partition von Dokumenten. Alle Dokumente die Teil dieser Partition sind, werden aus den anderen Possible-Root-Partitionen entfernt. Dies verhindert, dass Daten doppelt integriert werden. Ohne zusätzliche Informationen, wie z. B. Ontologien, ist es nicht möglich festzustellen, wo die Daten dieser Dokumente integriert werden soll. Die Integration erfolgt deshalb mit der Partition, für die als erste eine Einfügeposition gefunden wurde.
4	Einfügen von Possible-Root-Partitionen, für die nicht alle Daten eingefügt werden können Es wird untersucht, ob alle Daten der Partition eingefügt werden könnten, falls alle Dokumente der Partition Teil einer anderen Possible-Root-Partition A oder einer der aus den Possible-Root-Partitionen erstellten Partition von Dokumenten A wäre. Hierzu wird der in Tabelle 2 beschriebene Algorithmus in leicht modifizierter Form angewandt: Anstatt der Partition von Dokumenten wird die aktuell bearbeitete Possible-Root-Partition verwendet. Die Einfügeposition wird nicht im Grundgerüst des Unified View gesucht, sondern in der Partition A. Dabei kommen nicht nur die Blätter von Partition A, sondern auch alle anderen Knoten als Einfügepositionen in Frage. Hierdurch gehen möglicherweise einige strukturierenden Elemente der aktuell bearbeiteten Possible-Root-Partition verloren. Dies wird in Kauf genommen, da ein Dokument nicht zwei Partitionen zugeordnet werden kann und deshalb entweder die strukturierenden Elemente einer Partition verloren gehen oder Daten keine Einfügeposition besitzen und im Überlaufbereich eingefügt werden müssten. Beende Schritt 4. Können keine Daten der Partition ins Grundgerüst, aber Fragmente der Daten, wie in Abschnitt 1 dieses Schrittes beschrieben, in eine anderen Partition eingefügt werden, so füge die Dokumente der Possible-Root-Partition zu dieser Partition hinzu. Falls es mehrere Partitionen gibt, in die Dokumente eingefügt werden könnten, so füge sie in die erste gefundene Partition hinzu. Füge die Fragmente, welche nicht eingefügt werden konnten, zum Überlaufbereich hinzu. Beende Schritt 4. Kann nur ein Teil der Daten der Partition (echte Untermenge) ins Grundgerüst und können keine Fragmente der Daten in eine andere Partition eingefügt werden, so erstelle aus der Possible-Root-Partition eine Partition von Dokumenten und entferne sie aus der Menge der Possible-Root-Partitionen. Die Daten der Partition können in diesem Fall nicht vollständig eingefügt werden. Die entsprechenden Fragmente müssen in dem Überlaufbereich eingefügt werden.

3.9.5 Metadata-Repository

Die in den vorangegangenen Abschnitten vorgestellten Konzepte verwenden Beschreibungen bzw. Schemata für Dokumente, Partitionen und den Unified View. Auf konzeptueller Ebene dient das verwendete Beschreibungsmodell zwar nur der Illustration, ist aber tatsächlich auch gleichzeitig auf der Ebene der XML-Technologien ein abgeschlossenes und korrektes Modell für die Integration und damit auch für eine konkrete Realisierung, denn das XInclude-Verarbeitungsmodell beschreibt sehr genau die Auflösung der XInclude-Direktiven auf der Ebene des XML-Infoset, d. h., das den Unified View beschreibende Dokument mit den darin enthaltenen XInclude-Direktiven repräsentiert die vollständig materialisierte Sicht. Würde man das Dokument mit einem XInclude-fähigen DOM-Parser verarbeiten, so erhielte man tatsächlich eine Datenstruktur, in der die Gesamtheit der Daten vollständig enthalten wäre. Ein geeignetes Verarbeitungsmodell für ein verteiltes

Mediator-basiertes Integrationssystem über das Web ist dies allerdings nicht. Es muss ein geeigneterer Weg gefunden werden, um die Partitionierung, Integration und Anfrageverarbeitung in SHARX besser unterstützen zu können. Dies bedeutet zunächst die benötigten Struktur bzw. Schemainformationen in geeigneter Art und Weise zu verwalten. Ferner sollten die Informationen über Einfügepositionen und ihre Quellen mit den zugehörigen Pfaden für einen effizienten Zugriff nicht mittels XML-Technologien realisiert werden. Das Analysieren der XML-Dokumente ist sehr aufwändig und kostenintensiv. Vielmehr sollten entsprechende Speicherungsstrukturen verwendet werden [Tuc04]. Im Folgenden werden entsprechende Ansätze entwickelt und geeignete Speicherungsstrukturen vorgestellt.

Die Partitionierung der Dokumente, der Einfügevorgang in eine gemeinsame Sicht sowie die Vorbereitungen, um deskriptive Anfragen auf die gemeinsame Sicht zu ermöglichen, zählen zu den statischen Integrationsaspekten. In Abb. 3.20 sind der Verarbeitungsprozess und die damit verbundenen Daten aufgezeigt. Die Komponente Schemaextraktor ermittelt die Schemata der eingehenden Dokumente. Dabei wird für jede Dokumentklasse, wie zuvor beschrieben, ein gemeinsamer Schemagraph generiert. Während des Partitionierungsvorgangs werden vom Partition Miner Partitionen erzeugt. Diese beinhalten ebenfalls ein Schema, worin an bestimmten Stellen Verweise auf Dokumente mit Angabe eines Pfads im Dokument gespeichert werden. Dies entspricht den XInclude-Direktiven des Beschreibungsmodells.

Da es drei unterschiedliche Typen von Partitionen gibt, nämlich Partitionen von Dokumenten, Partitionen von Partitionen und die darin enthaltenen Possible-Root-Partitionen, muss jede Partition mit einer Typinformation versehen werden. Da die Dokumentklassen getrennt partitioniert werden, muss eine Partition auch Informationen darüber enthalten, welcher Dokumentklasse sie angehört. Handelt es sich um eine Partition von Partitionen so muss weiterhin vermerkt werden, welche Possible-Root-Partitionen sie umfasst. Der Einfügevorgang erzeugt die interne Repräsentation des Unified View. Weiterhin modifiziert er im Falle von Partitionen von Partitionen auch möglicherweise die darin enthaltenen Possible-Root-Partitionen. Zuletzt wird eine für die Anfrageverarbeitung geschaffene Version der internen Repräsentation des Unified View, hier Unified View Cache genannt, erzeugt.

Anforderungen

Es wurde bereits angemerkt, dass es sehr sinnvoll erscheint, Beschreibungsmodell und Schema in einer gemeinsamen Struktur zu verwalten. Diese Struktur sollte dauerhaft gespeichert werden, um nicht bei der Anfrageverarbeitung wieder beispielsweise eine Schemaextraktion oder Schematranslation durchführen zu müssen. Eine solche Struktur sollte ein um Verarbeitungsinformationen angereichertes Schema verarbeiten können, das die benötigten Zugriffspfade möglichst effizient unterstützt. Im Folgenden werden die Zugriffspfade für die einzelnen Komponenten, basierend auf den entwickelten Algorithmen zur Schemaextraktion, zur Partitionierung und dem Einfügen der Daten, entsprechend [Tuc04] analysiert.

Die Dokumentschemata werden vom Schemaextraktor erzeugt und anschließend vom Partition Miner weiter verarbeitet. Beim Erstellen des Schemas wird eine Navigation des Graphen anhand von Vater- und Kindbeziehungen durchgeführt. Es sollte daher möglich sein, alle Kinder eines Knotens

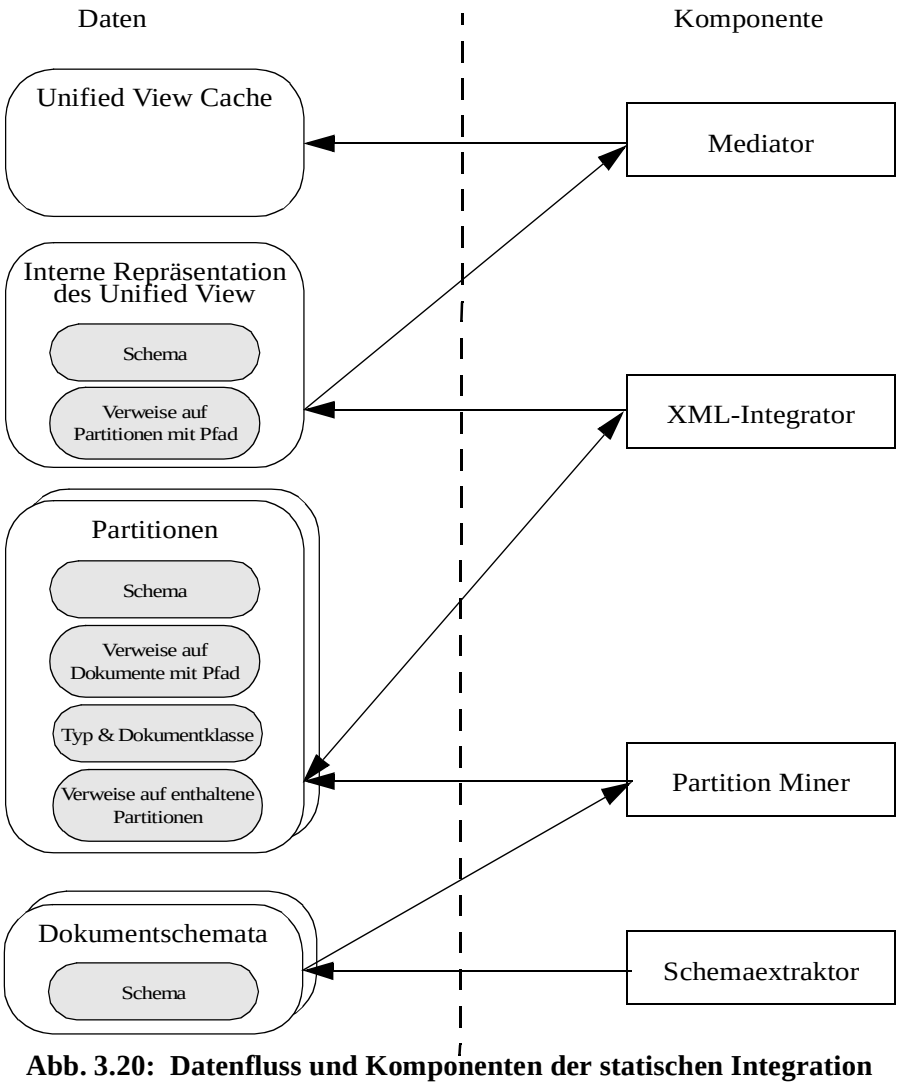


Abb. 3.20: Datenfluss und Komponenten der statischen Integration

sowie seine Vorgänger zu finden. Der Zugriff auf die Dokumentschemata erfolgt getrennt nach Dokumentklasse. Es sollten für den Vorgang der Partitionierung, gemäß des beschriebenen Vorgehens, gleiche Elemente, also solche mit gleichem Namen und Namespace-Definition, ausfindig gemacht werden.

Um unter Anderem feststellen zu können, ob es eine Einfügeposition gibt, sollte es weiterhin möglich sein das Kriterium der Pfadeindeutigkeit effizient zu überprüfen. Dies wird an mehreren Stellen der Algorithmen benötigt. Wie bereits beschrieben werden die Partitionen inkrementell erstellt und erweitert. Gemäß des beschriebenen Algorithmus zur Partitionierung sollte es daher möglich sein, im Schema über Vater- und Kindbeziehungen zu navigieren. Weiterhin wäre eine Möglichkeit zum einfachen Auffinden von gleichen Knoten einer Partition und eines Dokumentschemas von Vorteil, da dadurch die beschriebene Suche nach einem Integrationsknoten beschleunigt werden könnte. Für den Einfügevorgang sollten Partitionen nach ihrem Typ und ihrer Dokumentklasse auffindbar sein. Weiterhin muss es möglich sein, für eine Partition von Partitionen leicht die zugehörigen Possible-Root-Partitionen zu ermitteln.

Die interne Repräsentation des Unified View wird vom XML-Integrator aus der Partition der strukturierenden Dokumente erstellt. Auch sie besteht aus Schemainformationen, welche über Vater-

und Kindbeziehungen navigierbar sein sollten. Gemäß der beschriebenen Algorithmen zum Einfügevorgang sollten Knoten, die als Einfügeposition in Frage kommen, also Blätter der internen Repräsentation, leicht auffindbar sein. Weiterhin sollte es möglich sein, gleiche Knoten einer Partition und der internen Repräsentation aufzufinden.

Es fällt auf, dass viele Daten aus gleichen Bestandteilen zu bestehen scheinen: So enthalten alle Produkte Schemainformationen in Form von Graphen. Da diese Schemainformationen über die gleichen Inhalte, nämlich über eine feste Dokumentenbasis gebildet werden, kommen in den Partition- und Unified-View-Schemata – mit Ausnahme des Überlaufknotens – ausschließlich solche Knoten vor, welche schon in einem Dokumentenschema enthalten sind. Auch die notwendigen und optionalen Zugriffspfade auf die Daten enthalten viele Gemeinsamkeiten: So wird für jedes Produkt eine Möglichkeit zur Ermittlung der Vater- und Kindknoten eines Knotens gewünscht. Weiterhin wird das Auffinden von gleichen Knoten z. B. im Dokumentenschema und im Partitionsschemas oft benötigt. Für zwei der Komponenten muss auch ermittelt werden, ob ein Knoten als Integrationspunkt in Frage kommt, ob also ein Pfad zur Wurzel mit bestimmten Eigenschaften existiert.

Konzepte eines gemeinsamen Metamodells

Zur Umsetzung der Anforderungen bietet sich ein Modell an, in welchem die Knoten normalisiert gespeichert werden. Gemäß Definition sind Knoten, welche die gleiche Namespace-Definition und den gleiche lokalen Namen aufweisen, als identisch zu betrachten. Aus diesem Grund sollte auch jeder Knoten nur genau einmal gespeichert werden, unabhängig davon in welcher Beziehung bzw. welchem Schema er tatsächlich genutzt wird. Die Kanten des Schemagraphs sollten als Beziehung zwischen Knoten verstanden werden, wobei diese bidirektional navigierbar sein sollten, also in Richtung Vater-Kind und Kind-Vater. Da es sich bei dem Schema um einen Graphen handelt, sollen beide dieser Beziehungen die Kardinalität n:m haben. Ein solches Modell hat den Vorteil, dass nach Definition gleiche Knoten auch nur genau einmal gespeichert werden. Es deckt ebenso viele der Anforderungen an die Zugriffspfade ab: Eine Möglichkeit zur einfachen Ermittlung der Vater- und Kindknoten eines Knotens existiert. Weiterhin ist es auch leicht möglich herauszufinden, welche Knoten von verschiedenen Schemata bzw. Beschreibungen gleich sind. Ausgehend von einen Knoten können Vater- oder Kindbeziehungen gesucht werden, welche zu den gesuchten Schemata gehören. Werden welche gefunden, so ist der Knoten Teil des Schemas beider Produkte.

Das im vorherigen Abschnitt motivierte Metamodell für die Schemata der Dokumente, Partitionen und des Unified View lässt sich leicht um die in Abbildung Abb. 3.20 beschriebenen weiteren Daten ergänzen. Die Multiplizität eines Knotens könnte im Knoten selbst oder aber in den Beziehungen zwischen den Knoten vermerkt werden. Da die Multiplizität eines Knotens aber unabhängig davon ist, in welchem konkreten Schema der Knoten referenziert wird, empfiehlt sich die Speicherung im Knoten selbst. Bei Konflikten, also wenn der Knoten in einem Schema die Multiplizität 1 und in einem anderen die Multiplizität n hätte, wird diese immer auf n gesetzt. Die gleichen Möglichkeiten bestehen für den Typ des Knotens. Auch dieser könnte in den Beziehungen oder im Knoten abgelegt werden. Da die Knoten in den Schemata jedoch verschieden angeordnet sind und daher ein Knoten in mehreren Schemata an unterschiedlichen Stellen erscheinen kann, empfiehlt es

sich nicht, den Typ im Knoten selbst zu vermerken. Stattdessen ermöglicht die Speicherung in der Beziehung eine von der Anordnung abhängige Definition. Die notwendigen Verweise von unterschiedlichen Schemata auf die Erzeugnisse des vorherigen Schritts sind ebenfalls von der Position eines Knoten abhängig. Aus diesem Grund sollten auch sie an den Beziehungen gespeichert werden.

Wie zuvor erwähnt sollen in den Schemata der Partitionen Verweise auf Dokumente, ergänzt um einen XPath-Ausdruck, abgelegt werden können. Diese Verweise sollen jeweils an bestimmten Stellen im Schema der Partition abgelegt werden. Zur Umsetzung dieser Anforderung bietet es sich an, diese Verweise an den Beziehungen zwischen den Knoten als Quelle und den Dokumenten als Ziel zu speichern. Da an einer beliebigen Stelle in einem Dokumentschema beliebig viele Verweise auf andere Dokumente erfolgen können und ein Dokument von beliebig vielen Schemata referenziert werden kann, sollte diese neue Beziehung über die Kardinalität n:m verfügen. Sie sollte um ein Attribut ergänzt werden, das den XPath aufnehmen kann. Es müssen jedoch auch weitere sehr spezifische Informationen verwaltet werden können. So muss für eine Partition neben ihrem Schema auch ein Name, der Typ sowie die Dokumentklasse erfasst werden können. Um dies zu gewährleisten, muss es für ein Dokumentschema (inkl. Name, Dokumentklasse und Verweis auf das Wurzelelement des Schemas), für eine Partition (inkl. Name, Dokumentklasse, Verweis auf das Wurzelelements des Schemas, Partitionstyp und gegebenenfalls Verweise auf enthaltene Possible-Root-Partitionen) und für die interne Repräsentation des Unified View (inkl. Name und Verweis auf das Wurzelelement des Schemas) eine entsprechende Struktur geben. Für ein entsprechendes Informationsmodell, das auf dem Entity-Relationship-Modell basiert, bedeutet dies jeweils eine eigene Entität.

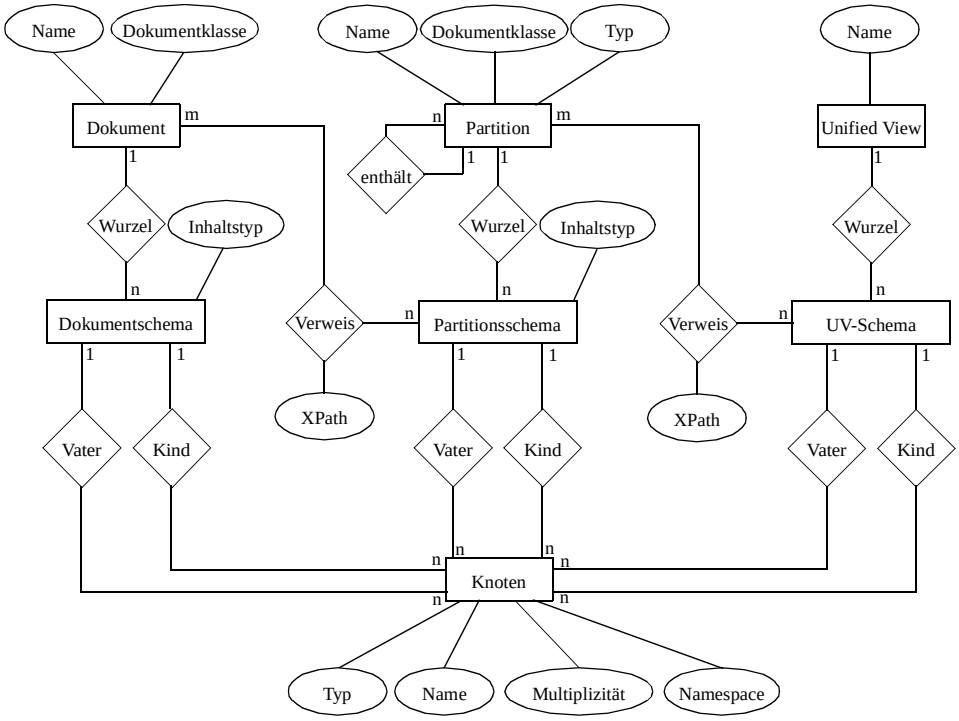


Abb. 3.21: Entity-Relationship-Diagramm des Metamodells

In Abb. 3.21 ist ein aus den Überlegungen in diesem Abschnitt resultierendes Entity-Relationship-Diagramm dargestellt. An die zentrale Entität „Knoten“ sind jeweils Strukturen zur Darstellung der Dokumente, der Partitionen und des Unified View gekoppelt. Diese ähneln sich und bilden drei vertikale Bereiche. Jeder dieser Bereiche besteht aus einer Schema-Entität, welche Kanten zwischen den Knoten darstellt und daher mit der „Knoten“-Entität in Beziehung steht, sowie aus einer Entität zur Repräsentation des Produkts selbst. Die Bereiche stehen auch untereinander mittels Beziehungen in Verbindung. Im Diagramm werden spezifische Informationen werden als Attribute der Entitäten „Dokument“, „Partition“ und „Unified View“ modelliert. Die spezifischen Beziehungen zwischen den Knoten sind mittels der „Dokumentschema“, „Partitionsschema“ und „UV-Schema“ Entitäten repräsentiert. Sie verfügen jeweils über zwei Beziehungen zur Knoten-Entität, welche Vater- und Kindknoten darstellen. Die Beziehung „Wurzel“ zwischen den produktspezifischen Informationen und den Schema-Informationen kennzeichnet die Beziehung zum Wurzelknoten des Schemas. Die Verweise vom Partitionsschema auf Dokumente und UV-Schema auf Partitionen sind über die Beziehung „Verweis“ abgebildet. An der Beziehung wird mittels des Attributs „XPath“ der XPath-Ausdruck des Verweises im Ziel vermerkt.

Die in Abbildung Abb. 3.21 dargestellten Attribute der Entität Knoten können sowohl Elemente als auch XML-Attribute aufnehmen: Durch das Attribut „Typ“ wird festgelegt, ob es sich bei dem Knoten um ein Element oder ein XML-Attribut handelt. Attributknoten könnten als Kinder von Elementknoten in das Schema eingefügt werden. Die Darstellung von XML-Attributen erlaubt es auch, auf dem ID/IDREF-Prinzip basierende Verweise in den Quelldokumenten zu unterstützen.

Durch das hier entwickelte Metamodell werden nicht alle in der Analyse geschilderten Zugriffspfade optimal unterstützt. Eine Aussage darüber, ob von einem bestimmten Knoten ein Pfad zur Wurzel, welcher ausschließlich über Knoten der Kardinalität 1 führt, existiert, lässt sich nur dadurch treffen, dass alle Pfade vom Knoten zur Wurzel des Dokuments über die Vater-Kind-Beziehungen ermittelt und überprüft werden. Dieser Aufwand ließe sich jedoch wesentlich durch eine geeignete Indexstruktur verringern: Dort könnte für jeden Knoten gespeichert werden, ob von ihm aus ein solcher Pfad existiert oder nicht: Falls ein Pfad existiert, so könnte er dort abgelegt werden. Eine vollständige Population des Index vor dem Partitionierungs- und Einfügevorgang würde jedoch dazu führen, dass zunächst für jeden im Schema des Produkts enthaltenen Knoten die Suche nach einem solchen Pfad durchgeführt werden müsste. Da der Index aber für jeden neuen Partitionierungsvorgang erneut erstellt werden muss, muss die für die Indexierung benötigte Zeit berücksichtigt werden. Der Vorgang der Erstellung des Index benötigt jedoch mehr Zeit, als durch dessen Nutzung gewonnen werden kann. Kommt beispielsweise nur ein Knoten als Integrationspunkt in Frage und muss dementsprechend nur für diesen einen Knoten ein Pfad zur Wurzel gesucht werden, so ist eine Suche ohne Index sicherlich weniger aufwändig, als zunächst zur Population des Index für jeden Knoten einen Pfad zur Wurzel zu suchen. Aus diesem Grund müsste der Index zur Laufzeit mit Informationen gefüllt werden und mehr eine Pufferungsfunktion besitzen. Muss für einen Knoten ein Pfad zur Wurzel ermittelt werden, so wird für diesen Knoten geprüft, ob im Index schon eine Aussage über ihn enthalten ist. Falls ja, so ist die Antwort gefunden. Falls nein, so werden die Vaterknoten ermittelt und die Suche bei diesen fortgesetzt. Gefundene Antworten – auch für alle Vaterknoten – werden bei Bedarf gepuffert und im Cache bzw. Index vermerkt.

3.10 Integration globaler Beziehungen

Beziehungen sind ein wichtiger Bestandteil des Gesamtkonzepts von SHARX. Sie erlauben eine Personalisierung des Unified View und ermöglichen es so jedem Benutzer, eine eigene Sicht auf den Unified View zu definieren. Im Folgenden wir das Beziehungskonzept von SHARX näher vorgestellt. Daraufhin wird analysiert, wie Beziehungen bei der Integration und der Ausführung von Anfragen unterstützt werden können und welche Arbeiten für eine Realisierung notwendig sind. Einige der anfallenden Aufgaben bei der Integration navigationsbezogener Dokumente können unter dem Gesichtpunkt der statischen Integrationsaspekte betrachten werden. Die Partitionierung und Integration navigationsbezogener Dokumente, das Bereitstellen von Informationen zur Generierung des Beziehungsdokuments und der Schemata sowie das Anpassen der internen Schemarepräsentation im Falle global spezifizierter Beziehungen sind solche Aufgaben, die vor der eigentlichen Anfrageverarbeitung durchgeführt werden können.

Verarbeitungsmodell

Grundsätzlich kann die Unterstützung von Beziehungen an verschiedenen Stellen in SHARX realisiert werden. Dies kann im Zug der Personalisierung und Anfrageverarbeitung geschehen oder physisch auf der Ebene der den Unified View implementierenden Komponente. Die Sichtbarkeit (lokal/global) von Beziehungen ist für diese Entscheidung sehr hilfreich. Eine Beziehung, die nur für einen Benutzer sichtbar sein darf, kann nicht auf Ebene des Unified View integriert werden. Dies würde bedeuten, dass sie vor allen anderen Benutzer verborgen werden muss. Global definierte Beziehungen hingegen können physisch auf der Ebene des Unfied View integriert werden. Hierbei ist besonders die Integration von deklarierten Beziehungen interessant, da entsprechende Referenzen bereits in den zu integrierenden Dokumenten vorhanden sind. Im Folgenden wird analysiert, in wie weit sich ein Zusammenhang zwischen der Integration von deklarierten Beziehungen mit globaler Sichtbarkeit und statischen Integrationsaspekten herstellen lässt.

Bei der Verarbeitung global deklarierter Beziehungen sind Referenzen zwar bereits in den zu integrierten Dokumenten vorhanden, allerdings müssen die Referenzen bei der Integration angepasst werden., insbesondere bei einer anderen Repräsentationsform als der ursprünglichen. Konkret ergeben sich zunächst folgende Aufgaben:

1. Sicherstellen der Eindeutigkeit wertbasierter Beziehungen im Unified View
2. Anpassung der Ziele bei der Integration
3. Transformation der zur Darstellung der Beziehung verwendeten XML-Technik in die als Repräsentationsform vorgesehene XML-Technik
4. Einbettung von Daten bei mittels XInclude repräsentierten Beziehungen.

Diese Punkte werden in [Tuc04] genauer untersucht und auch Möglichkeiten zur Verarbeitung diskutiert.

3.11 Anfrageverarbeitung

Die klassische Anfrageverarbeitung relationaler DBS trennt die Anfrageverarbeitung von der Anfrageausführung [Mit95]. Der logische DB-Prozessor bzw. Anfrageprozessor ist verantwortlich für die Anfrageverarbeitung. Das Ergebnis der Anfrageübersetzung ist ein optimierter Ausführungs-

plan, der durch den physischen DB-Prozessor bzw. das Anfrageevaluierungssystem mit den geringsten Kosten auf einer DB ausgeführt werden kann. Die Kosten werden dabei maßgeblich durch Externspeicherzugriffe bestimmt.

3.11.1 Phasen der klassischen Anfrageverarbeitung

Der Vorgang der Anfrageverarbeitung kann in mehrere Phasen unterteilt werden. Ausgehend von einer deskriptiven Anfrage wird zunächst eine Interndarstellung dieser Anfrage erstellt und sie auf syntaktische Korrektheit überprüft. Anschließend wird sie in eine standardisierte Darstellung überführt, um mittels geeigneter Umformungen die weitere Verarbeitung verbessern zu können. Dabei wird zunächst versucht, mögliche Verarbeitungsredundanzen zu erkennen und auf dem Hintergrund einer geeigneten Algebra das Anfrageverhalten zu optimieren. Nach dieser algebraischen Optimierung werden mögliche Ausführungspläne bestimmt und anhand eines geeigneten Kostenmodells der günstigste Plan ausgewählt, der anschließend zu Ausführung gelangt. In Abb. 3.22 ist dieser Ablauf grafisch dargestellt. Im ersten Schritt (Parser) wird die Anfrage nach der Übersetzung, Normalisierung und Vereinfachung in eine geeignete Interndarstellung überführt. Aus der Interndarstellung wird ein Anfragegraph erstellt, der zunächst optimiert und in einen Ausführungsplan (Translation) übersetzt wird. Die Translation kann auch als eine Optimierung angesehen werden, die aus einer Menge semantisch äquivalenter Ausführungspläne anhand eines Kostenmodells den günstigsten auswählt und zur Ausführung bringt. Dabei muss nicht unbedingt eine kostenbasierte Optimierung verwendet werden, die beispielsweise Statistiken in die Bewertung einbezieht, sondern es kann auch eine regelbasierte Optimierung gewählt werden.

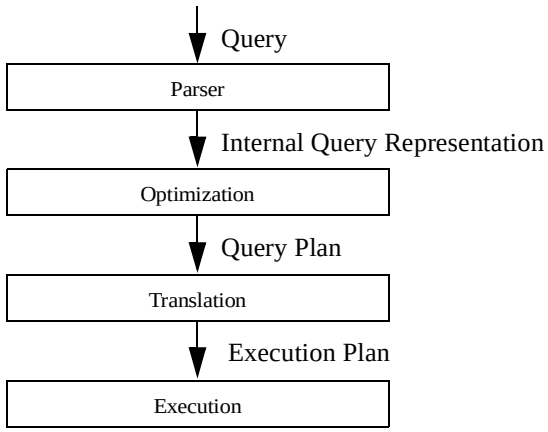


Abb. 3.22: Klassische Anfrageverarbeitung

In diesem Fall wird qualitativ der Einfluss semantikerhaltender Restrukturierungen (Heuristiken) auf das Anfrageverhalten abgeschätzt. Eine der wichtigsten Heuristiken besteht darin, die Größe der Zwischenergebnisse zu minimieren.Findet die Optimierung und Übersetzung im direkten Zusammenhang, beispielsweise bei so genannten Ad-hoc-Anfragen, mit der Anfrageausführung statt, so spricht man von dynamischer Optimierung. Während bei einer statischen Optimierung die Anfrage zunächst optimiert und übersetzt wird, um später möglichst mehrfach ausgeführt zu werden. Dies hat insbesondere bei Anfragen einen besonderen Vorteil, die aus Anwendungsprogrammen heraus auf die Datenbank zugreifen, sich in der Struktur nicht ändern und daher mehrfach ohne wiederholte Optimierung ausgeführt werden können. Von einer hybriden Optimierung spricht man, wenn nach einer Übersetzung weiterhin die Möglichkeit bestehen bleibt, zumindest einzelne Teile der Anfrage während Ausführung zu optimieren.

Verteilte und föderierte Ansätze

Der klassischen Anfrageverarbeitung liegt die Annahme zugrunde, die volle Kontrolle über ein zentrales Speicherungssystem zu besitzen. Aus Gründen der Leistungssteigerung kann es aber sinnvoll sein, Daten auf mehrere Knoten zu verteilen. Dies erfordert allerdings die Betrachtung von Strategien zur verteilten sowie parallelen Anfrageverarbeitung [Rah94], verspricht aber auch eine bessere Skalierbarkeit. Sind die einzelnen Knoten aber autonom und darüber hinaus auch heterogen, so spricht man bei entsprechenden Anfragesystemen von föderierten DB-Systemen. Beiden Ansätzen ist gemein, dass sie ein globales Schema präsentieren und den Aspekt der Verteilung und Heterogenität verbergen (Ortstransparenz bzw. Verteilungstransparenz). Auch verwenden sie in der Regel ein gemeinsames Datenmodell und eine gemeinsame Algebra. Bei fortschreitender Heterogenität und konzeptionell bedingter loser Kopplung spricht man hingegen von Integrationssystemen. Die Optimierung von Anfragen im verteilten Fall kann zentralisiert oder verteilt ausgeführt werden. Die zentralisierte Optimierung erfordert allerdings umfassendes globales Wissen, um eine verteilte Anfrage optimal ausführen können. Eine verteilte Optimierung kommt hingegen mit lokalem Wissen aus, was aber gleichzeitig einen erhöhten Kooperationsaufwand bedeutet. Eine hybride Mischform ist möglich. Die Anfrage wird dabei zuerst zentral vorgeplant und kann weiterhin lokal optimiert werden. Auf die Anforderungen an die Anfrageverarbeitung insbesondere von XML-basierten Integrationssystemen im Web soll im Folgenden näher eingegangen werden.

3.11.2 Anforderungen an eine Anfrageverarbeitung im Web

An eine verteilte Anfrageverarbeitung im Web werden andere Anforderungen gestellt als an eine klassische zentralistische Anfrageverarbeitung. Datenquellen weisen einen hohen Grad autonomen Verhaltens auf und sind sehr vielschichtig in ihren Fähigkeiten und Möglichkeiten. Ihre Funktionalität reicht von der Unterstützung deskriptiver Anfragesprachen bis hin zum alleinigen Bereitstellen von einfachen Dokumenten. Ferner können Anwortzeiten stark variieren und insbesondere Daten als Ströme auftreten. Die Anforderungen klassischer Systeme an die Ergebnismenge, korrekt und vollständig zu sein, macht im Falle Web-basierter Anfragesysteme wenig Sinn. Insbesondere vor dem Hintergrund wechselnder Verfügbarkeit ist die Zeit bis zur vollständigen Beantwortung einer Anfrage weniger wichtig geworden als die Zeitspanne, bis das erste Ergebnis zurückgeliefert werden kann [Bla+03]. Gerade im Umfeld Web-basierter Integrationssysteme, die mit den Problemen der Verteilung, dem hohen Grad an Autonomie und starker Heterogenität kämpfen, ist folglich der Ansatz einer klassischen Anfrageverarbeitung nicht anwendbar. Viele Web-basierte Anfragesprachen wurden entwickelt, Algebren entworfen und evaluiert [FLM98]. Ausgehend von der Adaption klassischer relationaler Ansätze [MMM97] über dynamische verteilte Anfrageverarbeitung [BKK+01, KW01] hin zu neueren Zero-Knowledge-Ansätzen, die sich durch adaptive und verteilte Anfrageverarbeitung und Optimierung auszeichnen, ist allen Ansätzen das Problem der Überwindung Web-immanenter Heterogenität und des ständigen Wandels des Web gemein. Eigenschaften der zu betrachtenden bzw. zu integrierenden Datenquellen ändern sich ständig, d. h. auch zur Laufzeit einer Anfrage. Sinnvolle aussagekräftige Statistiken oder Selektivitätsabschätzungen sind daher schwierig zu erhalten. Dynamische, flexible bzw. adaptive Ansätze zur Anfrageverarbeitung im Web sind gefordert, d. h. Ansätze, die mit Verteilung, Autonomie und Heterogenität derart umgehen, dass Veränderungen in lokalen Schemata, in Anwendungsschnittstellen durch entsprechend

flexible Wrapper, im Zusammenhang mit Verfügbarkeit keine Problemen darstellen und somit manuelle Eingriffe weitestgehend unnötig machen. Datenquellenbeschreibungen und Datenstrom-Orientierung sind Merkmale neuerer Web-basierter Integrations- bzw. Anfragesysteme.

3.11.3 Anfrageverarbeitung in SHARX

In SHARX wird zunächst ein klassischer Ansatz verfolgt, der an die Bedürfnisse XML-basierter Integrationssysteme im Web angepasst wurde und dessen Ausführungseinheit dynamisch genug ist, zur Verarbeitungszeit flexibel auf die ständigen Veränderungen im Web zu reagieren. Die Anforderungen an die Anfrageverarbeitung in SHARX bestehen darin, dass nicht nur statische XML-Dokumente, sondern auch dynamische XML-basierte Sichten auf Datenquellen (beispielsweise aus der Integration von DBS in das Web entstanden) unterstützt werden, deren Inhalt und Größe stark variieren können. Dies hat zur Konsequenz, dass Anfragen umgeschrieben werden müssen und Teilanfragen an die Datenquellen propagiert werden sollten. Es sollten nur die wirklich benötigten Daten angefordert und SHARX-seitig verarbeitet werden. Die Berücksichtigung der Integration der Daten und Beziehungen sollte Beachtung finden. Auch sollte die Menge der zu integrierenden Datenquellen zur Laufzeit dynamisch erweitert werden können. Diese Menge der so genannten registrierten Dokumente sollte analysiert werden und Ergebnisse in Form von Schemainformationen oder auch Statistiken in die Anfragerverarbeitung samt ihrer besonderen Eigenschaften und Fähigkeiten (Unterstützung von DOM/SAX) eingehen. Zur Erfüllung dieser Anforderungen wurde eine eigene, aber einfache Algebra entwickelt, die insbesondere die o. g. Eigenschaften berücksichtigt. Dabei wird zwischen logischen und physischen Operatoren unterschieden. Die Verwendung logischer Operatoren (siehe Abschnitt 3.11.5 „Operatoren“ auf Seite 59) im Anfrageplan erlaubt durch eine abstraktere Sicht auf die Anfrage und entsprechende semantikerhaltende Restrukturierungsregeln die Anfrage zu optimieren, ohne konkrete Implementierungsdetails betrachten zu müssen. Es können verschiedene Implementierungen für beispielsweise einen Join-Operator existieren, deren Wahl von Statistiken, Zugriffshäufigkeit oder Kardinalität der beteiligten Join-Partner abhängt. Diese physischen Operatoren werden während der Translationsphase ausgewählt (siehe Abschnitt „Translation: Erstellung eines Ausführungsplans“ auf Seite 69).

Mediator-basierte Anfrageverarbeitung

Die Anfrageverarbeitung basiert auf einer Kombination zweier Integrationsansätze: materialisierend und virtuell. Eine zu verarbeitende deskriptive Anfrage wird entsprechend der vorangegangenen Beschreibung der einzelnen Schritte einer klassischen Anfrageverarbeitung analysiert, optimiert, übersetzt und ausgeführt, wobei ein großes Optimierungsziel darin besteht, möglichst selektive Teilanfragen an die Datenquellen zu propagieren. Die Gesamtanfrage muss zu diesem Zweck in Teilanfragen zerlegt und die Teilergebnisse wieder zusammengeführt werden. Ergebnisse sollen aber für nachfolgende Anfragen wiederverwendet und daher zwischengespeichert werden. Die Anfragerverarbeitung bedient sich dabei der standardisierten deskriptiven Anfragesprache XQuery. Aus XQuery-Anfragen werden Teilanfragen in XPath abgeleitet und die Datenquellen angefragt. Die notwendigen Operatoren zur Beschreibung einer solchen Anfrage werden im Folgenden vorgestellt.

3.11.4 TSHARX – Mediator-Algebra

Die SHARX-Mediator-Algebra TSHARX stellt eine einfache Algebra mit mächtigen Operatoren dar: Projektion (Projection), Selektion (Selection), Zugriff (Access), Fusion, Verbund (Join), Sortierung (Sort) und einem Operator für die Aufbereitung des vorläufigen Ergebnisses (Tagger). Die Mächtigkeit der Anfragepläne, die mit diesen Operatoren gebildet werden können, ist allerdings beschränkt. Eine vollständige Unterstützung von XQuery ist durch diese Menge an Operatoren noch nicht möglich. Dazu fehlen beispielsweise Operatoren zur Beschreibung von Aggregationen (GroupBy) und Mengenoperationen (Intersect, Difference, OuterUnion, o. ä.). Allerdings kann bereits mit den o. g. Operatoren der Zugriff auf Datenquellen geplant und optimiert werden. Soll die vollständige Mächtigkeit von XQuery unterstützt werden, so kann die Operatormenge um spezielle Operatoren entsprechend ergänzt oder es kann auf die vorhandene Funktionalität aufgesetzt und ein mächtiger XQuery-Operator am Ende der Verarbeitung eingesetzt werden, der sich durch eine generische XQuery-Implementierung realisieren lässt (beispielsweise Lucents Galax[13], FHG-IPSIs IPSI-XQ[14] oder GNUs Qexo (Kawa-Query)[15]. Die ursprüngliche Anfrage würde zu diesem Zweck aufgespalten in einen Teil, der durch TSHARX beschrieben wird und eine Obermenge der zur Beantwortung der urpsrünglichen Anfrage benötigten Daten beinhaltet, und in einen Teil, der durch den generischen XQuery-Prozessor verarbeitet werden muss, d. h. aus dem ursprünglichen XQuery-Ausdruck muss eine Beschreibung der benötigten Daten extrahiert werden. Entsprechende Ansätzte finden sich beispielsweise in [MS03]. Die Daten können also derart aufbereitet werden, dass ein generischer XQuery-Prozessor auf dem Ergebnis der TSHARX-basierten Verarbeitung eine vollständige Unterstützung von XQuery realisieren könnte. Die dabei und im Allgemeinen verwendbaren Operatoren der TSHARX-Algebra werden im Folgenden vorgestellt.

3.11.5 Operatoren

Viele Vorschläge für eine XQuery-Algebra orientieren sich an der Relationenalgebra, um insbesondere vorhandene Optimierungskonzepte auf die XQuery-Anfrageverarbeitung übertragen zu können. XAL, beispielsweise, ist eine solche Algebra, die auch einen Unorder-Operator als einen besonderen Operator einführt, um die Anfrageverarbeitung besser optimieren zu können [FHP02]. In SHARX spielt die Reihenfolge der Daten eine sehr untergeordnete Rolle. Daher benötigt die SHARX-Mediator-Algebra TSHARX zur Optimierung im Zusammenhang mit Verbund-Operatoren keinen expliziten Unorder-Operator. Dies folgt bereits aus dem SHARX-Datenmodell und kann bei der Optimierung derart berücksichtigt werden, dass der Verbund-Operator kommutativ ist. Zwischen den Operatoren müssen Daten ausgetauscht werden können. XQuery definiert hierfür den Begriff der Sequenzen. In einer Sequenz können keine oder mehrere so genannter Items vorkommen. Items[16] können atomare Werte oder Knoten des XML Infosets repräsentieren, d. h., es können einfache Werte, Knoten oder ganze Teilbäume als Items in einer Sequenz vorkommen. Ein TSHARX-Operator erwartet eine Sequenz von Tupeln mit Items. Dies entspricht einem Tupel Stream in XQuery und auch weitestgehend dem in Niagara verwendeten Datenmodell [VGD+02].

13.) http://db.bell-labs.com/galax/

14.) http://ipsi.fhg.de/oasys/projects/ipsi-xq/

15.) http://www.qexo.org/

16.) Für Item gibt es unserer Meinung nach leider keine sinnvolle deutsche Übersetzung [Bue03], da die treffende Übersetzung Element doch gerade im XML-Bereich sehr überstrapaziert ist.

Da Operationen direkt auf den (Teil-)Bäumen (Trees) ausgeführt werden können und keine Konvertierung oder weitere Materialisierung notwendig ist, wurde analog zu der „Tree Algebra for XML (TAX)“ [JLST01] für die SHARX-Mediator-Algebra der Name TSHARX ausgewählt. Die von ihm erwartete Mindestlänge eines Tupel ist abhängig von der Anzahl der gebundenen Variabeln, d. h., ein Item-erzeugender Operator, z. B. der Access-Operator zum Zugriff auf eine Datenquelle, erzeugt eine Sequenz von Tupeln der Länge 1, ein einziger binärer Join-Operator erzeugt Tupel der Länge 2 und jeder weitere Join-Operator erzeugt Tupel der Länge n+1. Einem zwei Join-Operatoren nachfolgenden Selection-Operator müssen beispielsweise für die Auswertung des Prädikats

($a/id = $b/id and $b/id = $c/id)

Tripel zur Verfügung gestellt werden. Da Tupel geordnet sind, können Variabeln an ein bestimmtes Element einer bestimmten Position im Tupel gebunden werden. Operatoren arbeiten auf diesen Items (beispielsweise Selection) bzw. erzeugen sie (beispielsweise Access, Projection). TSHARX sind insgesamt folgende Operatoren bekannt:

- Access
- Fusion
- Projection
- Selection
- Convert (Adaptor)
- Unnest
- Join
- Union
- Sort
- Tagger (Collector).

Access

Der Zugriff auf ein Dokument und damit auf eine Datenquelle wird durch den Access-Operator symbolisiert. Der Access-Operator benötigt dazu mehrere Parameter: URI, Access Method und XPath. Der Uniform Resource Identifier (URI) [RFC2396] lokalisiert das Ziel eines SHARX-internen Zugriff auf eine Datenquelle mittels eines Data Provider. Die Angabe der Access-Method legt fest, ob die Daten Stream-basiert (SAX), DOM-basiert oder als Text geholt werden sollen. XPath ist ein optionaler XPath-Ausdruck, der datenquellenseitig ausgewertet werden soll.

Fusion

Der Fusion-Operator repräsentiert die Integration der Daten aus den verschiedenen Datenquellen und symbolisiert somit gleichzeitig den Unified View. Die jeweiligen Daten werden durch diesen Operator in die Strukturen abgebildet, die durch die Partitionierung und XML-Integration festgelegt wurden. Dabei werden auch Beziehungen berücksichtigt: also die Abbildung lokaler Schemata auf globale und die Integration global spezifizierter Beziehungen.

Projection

Der Projection-Operator in TSHARX unterscheidet sich ein wenig von dem relationalen Projektionsoperator. Er beschreibt durch eine Anzahl von XPath-Ausdrücken den für die Anfrageverarbeitung relevanten Teil eines Dokument und schneidet (oder „projiziert“) diesen aus dem Dokument heraus. Der Projection-Operator stellt eine Verallgemeinerung des in anderen XML-Query-Algebren häufig vorkommenden Navigation-Operators dar [ZR02]. Während Navigation-Operatoren in der Regel nur einen Pfad-Ausdrucken auswerten (d. h. navigieren) und das Ergebnis dieser Navigation zurückliefern, kann der TSHARX-Projection-Operator analog zu GALAX [FSC+03] ganze Fragmente, also mehrere Teilbäume unter Beibehaltung der ursprünglichen Struktur eines Dokuments, auswählen. Dies geschieht beispielsweise durch die Anwendung der in [MS03] beschriebenen Verfahren unter Berücksichtigung der Pfadausdrücke der ursprünglichen XQuery-Anfrage und der vorhandenen Schemainformationen. So können auch aus Pfaden wie beispielsweise //a/*/b konkrete Anfragen abgeleitet werden. Einfache Selektivitätsabschätzung auf der Basis des globalen Schemas [AAN01] dienen in SHARX der Entscheidung, ob ein Dokument vollständig geholt (Data Shipping) oder die Anfrage möglichst nah bei der Datenquelle ausgewertet werden sollte (Query Shipping). Der Projection-Operator kennt nur Pfade als Parameter und keine vollständigen XPath-Ausdrücke in Verbindung mit Prädikaten. Daher korrespondiert er auch gleichzeitig mit der FOR-Klausel einer XQuery-Anfrage in der SHARX-eigenen Implementierung und steht somit in der Regel direkt vor einem Access- oder Fusion-Operator. Seine Position und letztlich auch die Tatsache, ob durch ihn eine Variable gebunden wird, entscheidet über das Ausgabeformat. Vor einem Access-Operator und damit unterhalb eines Fusion-Operators liefert er genau ein Item zurück, da unterhalb des Fusionsoperators noch keine Variablen gebunden sein können. Anhand der Pfadausdrücke wird aus einem Dokument der für die Beantwortung einer Anfrage relevante Teil bestimmt und unter Beibehaltung der Gesamtstruktur des Dokuments heraus projiziert. Oberhalb des Fusionsoperators bietet er die Funktionalität eines stark vereinfachten Selection-Operators, liefert eine Sequenz von Tupeln zurück und bindet diese an eine Variable, wodurch sich die Beziehung zu der FOR-Klausel erklärt. Der Projection-Operator bietet somit die Flexibilität das Ergebnis einer Projektion je nach Anforderung in unterschiedlichen Ausgabeformaten anzubieten und behält dabei die Möglichkeit, einfache Pfadanfragen gegenüber einer allgemeinen und damit auch sehr komplexen XPath-Anfrage hoch effizient und optimiert ausführen zu können.

Selection

Der Selection-Operator entspricht von der Bedeutung und Anwendung einem Selektionsoperator im relationalen Modell. Allerdings können Selection-Operatoren in TSHARX vollständige XPath-Ausdrücke verwenden. Sie werden auf eine Menge von Items bzw. Sequenzen angewendet und schränken die Menge der vorkommenden Items auf die sich qualifizierenden ein. Aufgrund von Optimierungsüberlegungen unterscheiden sich aber die Semantik des Selektionsoperators in der Verwendung vor bzw. nach dem Fusion-Operator. Unterhalb des Fusion-Operators ist die Selektion nur als Hinweis zu verstehen, d. h., sie stellt keine scharfe Bedingung für die Spezifikation der benötigten Daten dar, sondern sie dient der Optimierung des Zugriffes auf die Datenquellen. Der Einsatz des Selection-Operators unterhalb des Fusion-Operators kann dazu verwendet werden, um bei einem Zugriff auf eine Datenquelle diese Bedingung mit an den Access-Operator zu binden. Zur

Minimierung der Zwischenergebnisse kann die entsprechende Ausführungseinheit den Selection-Operator ausführen oder aus Gründen der Zwischenspeicherung (Caching) von Daten aber auch ignorieren. Die unterhalb eines Fusion-Operators verwendete Selektionsbedingung kann eine Abschwächung der oberhalb aus einer Anfrage abgeleiteten Selektionsanforderung sein. Dies bedeutet aber auch, dass oberhalb des Fusion-Operators eine Bedingung entsprechend der Anfrage erhalten bleibt und nicht umgeschrieben wird. Dies kann zur Folge haben, dass eine Bedingung mehrfach ausgewertet werden muss. Allerdings entspricht dies genau der Sichtweise, die mit dem Fusion-Operator verbunden ist. Der Fusion-Operator symbolisiert den gesamten Unified View, so dass folglich oberhalb dieses Operators alle entsprechenden Restriktionen, Restrukturierungen oder Sortierungen ausgeführt werden müssen.

Unnest

Items können nicht nur atomare Werte und einfache Knoten repräsentieren, sondern es können auch Teilbäume unterhalb der Knoten existieren. Anfragen können sich auf Kindelemente dieser Bäume beziehen, beispielsweise in folgender Anfrage:

for $a in /a, b$ in $a/b
return {$a, $b}

Diese Anfrage ist nicht äquivalent zu der folgenden Anfrage:

for $a in /a, b$ in /a/b
return {$a, $b}

Ein entsprechender Anfrageplan kann also nicht so aussehen, dass zweimal auf dieselbe Datenquelle zugegriffen wird, sondern es muss ein Aufbrechen der unterhalb von a liegenden Strukturen erfolgen. Diese Unnest-Operation gleicht im Ergebnis einem Join, allerdings besitzt sie nur genau eine eingehende Kante. Aber sie kann weitere Variablen binden, beispielsweise $b bei oben dargestellter Anfrage. Im Allgemeinen kann die Reihenfolge der Ausdrücke in der For-Klausel einer XQuery nicht vernachlässigt werden, da ein Unnest aber auch als besonderer Join-Operator angesehen werden kann und Join-Operatoren kommutativ sind, kann auch hier die Reihenfolge unbeachtet bleiben. Das hat den Vorteil, dass mehrere von einer gemeinsamen Variable abhängigen Variablen auch gemeinsam in einem Unnest-Operator verarbeitet und gebunden werden können. Betrachtet man dazu die For-Klausel einer Xquery-Anfrage:

for $a in /a, $b in /b, $c in $a/c, $d in $b/d,

so bedeutet dies, dass zuerst $a und $c über einen Unnest-Operator verarbeitet und gebunden werden, dann ein Join-Operator für $b folgt und zuletzt wieder ein Unnest-Operator für die Bindung von $d. Dies würde eine treppenartige Kette von Unnest- und Join-Operatoren ergeben, da aber auch Unnest und Join kommutativ sind, kann aus dieser Kette durch entsprechende Umformungsregeln auch ein Baum erzeugt werden, bei dem zuerst die Unnest-Operatoren und dann der Join ausgeführt würde. Entsprechende Optimierungsregeln dazu findet man im Zusammenhang mit dem Path-Operator in YAT [SA02].

Union, Join, Sort

Union, Join- und Sort-Operatoren findet man nur oberhalb bzw. nach einem Fusion-Operators. Sie entsprechen weitestgehend der bereits bekannten Semantik der relationalen Algebra. Allerdings müssen Datenströme, die über den Union-Operator vereinigt werden, nicht, wie im relationalen Fall, vereinigungsverträglich sein.

Tagger (Collector)

XQuery bietet die Möglichkeit der Restrukturierung. Für den einfachsten Fall bedeutet dies, dass neue Elemente zu erzeugen sind, die weitere Strukturen um das Anfrageergebnisse herum aufbauen. Das Erzeugen neuer Elemente übernimmt der Tagger-Operator. Dieser Operator erwartet eine Sequenz von Tupeln, deren Länge innerhalb einer Anfrage (FLOWR-Ausdruck bzw. Return-Klausel) durch die Anzahl der gebundenen Variabeln bestimmt wird. Nach der vollständigen Verarbeitung eines FLOWR-Ausdrucks können allerdings weitere strukturierende Elemente hinzuzufügen sein. Da keine Variablen mehr gebunden sind, erwartet der Tagger-Operator Tupel der Länge 1 und fügt sie an der entsprechenden Stelle ein. Diese Stelle ist eindeutig durch die Position des FLOWR-Ausdrucks in der Anfrage bestimmt. Im Falle mehrerer nicht verschachtelter FLOWR-Ausdrücke innerhalb einer gemeinsamen Anfrage, kann das Aufsammeln der einzelnen Ergebnisströme bei nur einer eingehenden Kante beliebig aufwändig werden. Daher wird anstelle eines aufwändigen Tagger- / Union-Konstrukts, wie in [Lut04] beschrieben, ein spezieller Tagger-Operator verwendet, der alle Datenströme aufsammelt und zu einer endgültigen Struktur verschmelzt. Dieser so genannte Collector-Operator ist ein n-ärer Operator und hat beliebig viele eingehende Kanten. Eingehende Sequenzen besitzen die Tupellänge 1 und sind an keine Variablen mehr gebunden.

Convert (Adaptor)

Der Convert-Operator symbolisiert die explizite Umformung ausgehend von einem Format auf ein anderes und kann notwendig sein, wenn die Möglichkeiten bzw. Fähigkeiten einer Datenquelle etwas anderes nicht zulassen. Ferner kann dies zu Optimierungszwecken geschehen. Desweiteren ist ein Adaptor-Operator vorgesehen, der die Kompensationseigenschaften der Adaptor zum Zugriff auf die Datenquellen steuert. So lässt sich zentral die Optimierung für alle leistungsbestimmenden Merkmale steuern. Dies schließt besonders die unterschiedlichen Fähigkeiten und Möglichkeiten der verschiedenen Datenquellen ein.

3.11.6 Anfrageplan und Ausführungsplan

In einem als Graph dargestellten Anfrageplan symbolisieren die Knoten Operatoren auf den Daten, die wiederum durch die Verbindungskanten repräsentiert werden. Die Daten[17] entlang der Kanten können aber ganz unterschiedlich repräsentiert werden, d. h., die Ein- und Ausgaben der Operatoren können ganz unterschiedliche Formate (DOM, SAX, Text) unterstützen bzw. verwenden. Dies entspricht den unterstützten Formaten der Data Provider und unterscheidet gleichzeitig einen Anfrageplan von einem Ausführungsplan. Der Anfrageplan symbolisiert eine Beschreibung des logischen Zugriffs auf die Datenquellen zur Ausführung einer Anfrage, während der Ausführungsplan

17.) Daten sind durch unterschiedliche Formate, d. h. durch unterschiedliche Datenmodelle wie DOM oder SAX, repräsentierte Items bzw. Sequences. Die Begriffe Item bzw. Sequence entstammen der XQuery-Spezifikation.

eine Konkretisierung darstellt. Der Ausführungsplan stellt eine Konkretisierung des Anfrageplans dar und ergänzt diesen um konkrete Implementierungshinweise der jeweiligen Operatoren unter Berücksichtigung der jeweiligen Data-Provider-spezifischen Attribute. Dies kann die Wahl eines bestimmten Formats bzw. der zu verwendenden Schnittstelle (DOM, SAX oder Text), des einzusetzenden Adaptors oder des Caching-Verhaltens betreffen (STATIC/PRELOAD oder VIRTUAL/(NOT-)VARYING). Die Optimierung des Anfrageplan zielt hauptsächlich darauf ab, möglichst alle sinnvollerweise bei einer Datenquelle auszuführenden Operationen auch tatsächlich entsprechend der Fähigkeiten der Datenquelle dort auszuführen. Das schließt beispielsweise die Verwendung unterschiedlicher XPath-Implementierung für die Selektion (SAX oder DOM-basiert) ein. Dieser Ansatz stellt folglich eine flexible Unterstützung von Stream-basierten (SAX), Proxy-basierten (DOM) und materialisierenden bzw. generischen Ansätzen (Text) dar. Proxy- oder Stellvertreterobjekte können im Falle von DOM eingesetzt werden, um die Ergebnisse, die durch einen Data Provider zur Verfügung gestellt werden, nicht importieren bzw. kopieren und damit auch materialisieren zu müssen, sondern den Zugriff an einzelne Objekte der darunter liegenden DOM-Schnittstelle des Data Provider propagieren zu können. Bei Stream-basierten Ansätze wäre ein Scheduler wichtig bzw. notwendig, um die Synchronisation der einzelnen Datenströme zu optimieren.

Beispielhafte Anfrageverarbeitung

```
<results> {
   for      $a in /uv/article,
            $p in /uv/person
   where $a/year > "2000"
   and $a/author/name = $p/name
   return
        <article>
            <a>{$a/title}</a>
            <p>{$p}</p>
        </article> }
</results>
```

Abb. 3.23: Beispielanfrage

Die in Abb. 3.23 gezeigte Anfrage soll im Folgenden als Beispiel dienen. Die Anfrage soll Aufsätze (Article) ab einem bestimmten Erscheinungsjahr aufzählen und alle verfügbaren Daten der jeweiligen Autoren (Person) hinzufügen. Die für XQuery-Anfrage typische Grundstruktur ist deutlich erkennbar: In der For-Klausel werden Pfade bzw. die durch diesen Pfadausdruck adressierten Knoten im Unified View als Sequenz an eine Variablen gebunden. Der Anfragekontext ist durch den Unified View gegeben und muss nicht noch einmal zusätzlich spezifiziert werden. Dann wird über die Knoten der Sequenz iteriert und das in der Where-Klausel angegebene Prädikat überprüft. Für alle sich qualifizierenden Knoten wird die Return-Klausel ausgewertet. Die Bindung von zwei Variablen im Beispiel zeigt die Notwendigkeit eines Verbundoperators. Im Zuge der Anfrageverarbeitung wird zunächst die Anfrage übersetzt, die Syntax geprüft sowie anschließend ein erster Anfrageplan generiert. Zu diesem Zweck wird aus der Interdarstellung ein Anfrageplan entsprechend der Darstellung als Graph in Abb. 3.24 generiert.

Bei dieser Generierung wird zunächst mit Hilfe der internen Schemarepräsentation die Menge aller zu betrachtenden Dokumente der jeweiligen Datenquellen bestimmt. Da alle Datenquellen mit ihren Dokumenten zuvor registriert worden sind und Informationen über alle vorkommenden Elemente gesammelt wurden, ist die Auswahl ohne weiteres möglich. Für jedes Dokument wird ein entsprechender Access-Operator dem Anfrageplan hinzugefügt. Prinzipiell kann der Access-Operator auch auf eine Sammlung (Collection) von gleichartigen Dokumenten angewendet werden. Konzeptionell werden dann alle beteiligten Dokumente durch einen Fusion-Operator integriert und ste-

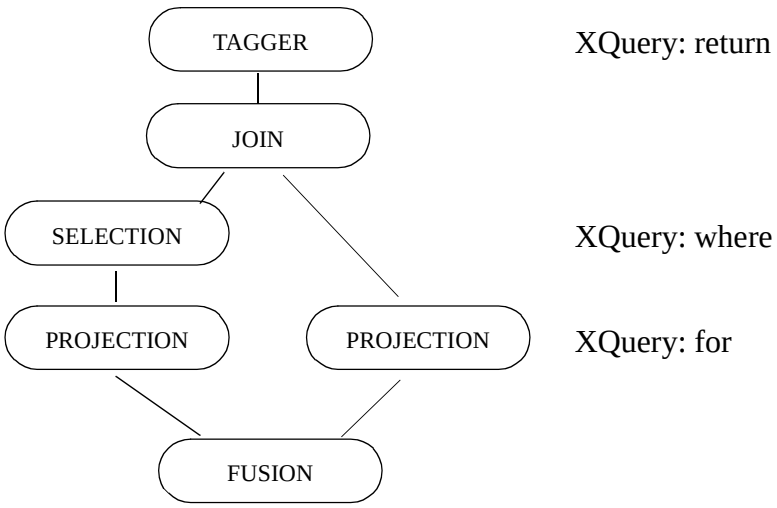


Abb. 3.24: Beispielanfrageplan

hen für die weitere Anfrageverarbeitung entsprechend zur Verfügung. Die Bindung der Variable „a“ und „p“ im Beispiel korrespondiert mit den zwei Projection-Operatoren oberhalb des Fusion-Operators in Abb. 3.24. Im linken Zweig wird das mit der Variable „a“ verbundene Prädikat (year > "2000") ausgewertet, bevor es zusammen mit dem rechten Zweig in einem Join-Operator kombiniert wird. Die Platzierung der Selection-Operatoren unterhalb des Join-Operators ist kein Ergebnis einer Optimierung, sondern Komsequenz des unterstellten Algorithmus, dem die Heuristik zugrunde liegt, die Selektion möglichst früh auszuführen (Selection Push Down). Diese Heuristik wird auch bei der eigentlichen Optimierung des Zugriffes auf die Datenquellen angewendet. Bei der Zerlegung der Anfrage und Erstellung des Anfrageplans werden letztlich Abbildungen auf XPath-Ausdrücke vorgenommen und die über XPath hinausgehenden Konzepte nachgebildet. Die Nachbildung erreicht nicht, wie bereits angedeutet, den vollen Sprachumfang der XQuery-Spezifikation, erlaubt allerdings die Optimierung des Zugriffs auf die Datenquellen.

Die Anfrage kann nun hinsichtlich vieler Faktoren optimiert werden. Die Anfrage (bzw. die zugehörigen XPath-Ausdrücke) kann weiterzerlegt werden, um die Anfrage möglichst nah bei der Datenquelle ausführen zu können. Oder es kann auch versucht werden, die Anzahl der entfernten Zugriffe zu minimieren. darüber hinaus kann prinzipiell noch zwischen strombasiertem und materialisiertem Zugriff entschieden werden. Bei der Optimierung müssen die Fähigkeiten und besonderen Eigenschaften der jeweiligen Datenquellen berücksichtig werden.

Optimierung

Die Optimierung einer Anfrage in SHARX, respektive eines Anfrageplans, hat primär nicht die Optimierung der eigentlichen Anfrage zum Ziel, sondern die Optimierung des Zugriffsverhalten auf die jeweiligen Datenquellen. Das bedeutet einerseits die Anzahl der entfernten Datenquellenaufrufe zu minimieren und andererseits auch möglichst viel der ursprünglichen Anfrage zu delegieren.

Optimierungsziele

Die Optimierung einer SHARX-Anfrage setzt an verschiedenen Punkten an. Die Optimierung eines Anfrageplans versucht möglichst viel der ursprünglichen Anfrage zu delegieren und Zwischenergebnisse zu minimieren. Dies geschieht einerseits durch Selektivitätsabschätzung, Konstantenpropagierung und durch Restrukturierung des Anfrageplans (Prädikatenzerlegung, Selection Push Down und Zusamenfassen einfacher Selektionen und Zusammenfassen von Selektion und Access-Operator). Andererseits werden bei der Anfragetransformation und Erstellung des Ausführungsplans die besonderen Eigenschaften der Datenquellen berücksichtigt und der Anfrageplan um interne Hinweise ergänzt, die Aufschluss über die verwendeten Adaptoren und das verwendete Datenformat (DOM, SAX, Text) geben und Hinweise auf konkrete Implementierungen der jeweiligen Operatoren liefern. So können zentral alle leistungsbeeinflussenden Faktoren kontrolliert und adaptiv auf die möglichen XML-Wrapper, d. h. konkret XMLDB:API-Implementierungen, eingegangen werden.

Betrachtung der in Frage kommenden Datenquellen

Über alle Stufen der Registrierung, Schema-Extraktion, Partitionierung und Integration hinweg werden kontinuierlich Daten über die Eigenschaften der Dokumente und die Fähigkeiten ihrer zugehörigen Datenquellen gesammelt und im Repositorium abgelegt. Anhand dieser Daten werden die bei einer Anfrage zu betrachtenden Dokumente ausgewählt. Zu diesem Zweck werden alle bereits extrahierten Pfade aus den Join-, Projection- und Selection-Operatoren betrachtet und die Menge aller vorkommenden XML-Elemente und -Attribute gebildet. Aus dieser Menge wird mit Hilfe des Repositoriums die Menge der zu betrachtenden Partitionen und anschließend der Dokumente abgeleitet. Ist diese Menge leer oder besteht sie nur aus Asterisken (*), so müssen alle Dokumente betrachtet werden.

Projektion und Restrukturierung (Selection Push Down)

Für die an der Beantwortung einer Anfrage beteiligten Datenquellen bzw. ihrer Dokumente kann aus der Anfrage ein Projektionspfad gebildet werden Aus der Beispielanfrage in Abb. 3.23 lassen sich nach [MS03] folgende Pfade ableiten:

- /uv/article,
- /uv/person (#),
- /uv/article/year,
- /uv/article/author/name,
- /uv/person/name,
- /uv/article/title (#).

Die Pfade, die mit einem Hash-Zeichen (#) gekennzeichnet sind, markieren Projektionspfade, die in der Return-Klausel vorkommen und für die Ergebnisaufbereitung benötigt werden. Ferner lässt sich ein Selektionskriterium ableiten:

- /uv/article/year > "2000".

Diese Pfade müssen nun gemäß der lokalen Schemata umgeschrieben werden und im Anfrageplan vor diejenigen Access-Operatoren eingefügt werden, deren adressierten Dokumente die jeweiligen Elemente beinhalten. Allerdings kann nur ein Teil der Projektionspfade sinnvoll als XPath-Anfrage nah bei der Datenquelle ausgeführt werden, da XPath in der Version 1.0 die Projektion nicht direkt unterstützt. Erst die Version 2.0 berücksichtigt diese Operation. Einzelne Pfade und Prädikate lassen sich aber sehr wohl auch in ihrer Gesamtheit als XPath-Anfrage an die Datenquelle delegieren. Die Ergebnisse müssen dann allerdings entsprechend der Anforderungen aufbereitet werden, d. h. evtl. Zusammensetzen der Teilergebnisse zu einem Dokument, um die geforderte Funktionalität der Projektion zu realisieren.

Betrachtung der Selektivität

Die Minimierung der Zwischenergebnisse ist ein ganz wichtiges Optimierungsziel. Allerdings kann die Auswertung des delegierten XPath-Ausdruckes sehr aufwändig sein und sich trotzdem ein großer Anteil des Dokuments qualifizieren. Hier ist ein Kompromiss zu finden zwischen der Komplexität des delegierten XPath-Ausdrucks, der Größe des Dokuments und der Größe des zu erwartenden Anfrageergebnisses. Allerdings erfordert insbesondere die Abschätzung der Größe des zu erwartenden Anfrageergebnisses detaillierte Kenntnisse über die Struktur des Dokuments und die Häufigkeit des Vorkommens einzelner XML-Elemente bzw. Attribute. Solche Informationen liefern im Allgemeinen spezielle Statistiken, beispielsweise wie [AAN01] vorgeschlagen. Datenquellen liefern aber von sich aus leider keine solchen detaillierten Daten, die man zum Aufbau von Statistiken verwenden könnte. Zwar werden beispielsweise in dem DISCO-Ansatz [NGT98] auch von den Wrappern die Bereitstellung von statistischen Informationen bzw. Daten zur Kostenbestimmung gefordert, doch ist das zugrunde liegende Datenmodell nicht so komplex wie das hierarchische Datenmodell von XML und den zugehörigen Spezifikationen (insbesondere XPath). Es liegt ein einfaches Objektmodell zugrunde und nur einfache statistische Daten werden betrachtet (Kardinalität der Objektmenge, durchschnittliche Größe eines Objekts, Anzahl der unterschiedlichen Ausprägungen oder Minimum und Maximum eines jeden Attributs). Solche Größen lassen sich in diesem Fall sehr einfach bestimmen, doch lässt sich dieser Ansatz auch nur sehr schwer auf XML und Xpath übertragen. Bietet dessen Datenmodell doch zu viele Freiheitsgrade. Möglich sind auch kalibrierende Ansätze, in denen aus Beobachtungen während der Anfrageausführung statistische Daten gewonnen werden (Histogramme), indem die Größe eines Anfrageergebnisses in Relation zu einem entsprechenden Prädikat gebracht wird. Diese Selektivitätsabschätzungen werden laufend präzisiert und in einem generischen Kostenmodell einer zentralen Optimierungskomponente evaluiert. Auch dieses Vorgehen ist aber aufgrund der Komplexität der XPath-Ausdrücke höchstens für sehr einfache Pfad-Ausdrücke und Prädikate vorstellbar. In SHARX wird daher die Menge der zu betrachtenden Operatoren auf die Vergleichsoperatoren (eq, ne, lt, le, gt, ge) für Werte (Value Comparison Expression) eingeschränkt und zunächst mit Konstanten für die Abschätzung der Selektivität nach [Mit95] gearbeitet.

Die Selektivität eines einfachen Pfadausdrucks in XPath kann anhand der Schemainformationen und weiteren statistischen Daten nach [AAN01] bestimmt werden, indem für jeden Elementtyp des Schemas die absolute Häufigkeit berechnet wird, mit der ein Element tatsächlich in der Menge der Dokumente auftritt. Da solche Daten aber aus den gleichen schon bereits genannten Gründen nur

sehr schwer zu berechnen sind und dies eine vollständige Analyse aller Dokumente der Dokumentenbasis erfordern würde, wird in SHARX zunächst nur eine sehr viel gröbere Abschätzung allein anhand der Schemainformationen durchgeführt. Während der Anfrageverarbeitung könnten dann exaktere Daten sukzessive aus den Anfrageergebnissen gewonnen werden (Histogramme).

In SHARX wird allerdings davon ausgegangen, dass die Daten über alle Knoten gleichverteilt sind. Es wird ein Element in Beziehung gesetzt zu der Anzahl der Geschwister, um die Selektivität eines Pfades abzuschätzen. Dabei wird zwischen einmaligem und mehrmaligem Auftreten unterschieden. Betrachtet man beispielsweise das Wurzelelement aus obigem Beispiel und setzt es in Beziehung zu den Geschwistern, so erhält man trivialerweise eine Selektivität von 1, da das Wurzelelement keine Geschwister besitzt. Die Selektivität eines Kindelements zu bestimmen ist interessanter. Bei einem Kindelement wird die Anzahl der einfachen und mehrfachen Elementtypen bestimmt. Ist die Anzahl der mehrfach vorkommenden Elementtypen größer als Null, so dominieren die mehrfach vorkommenden Elementtypen die einfachen Elementtypen. Einfache Elementtypen werden nur betrachtet, wenn unter den Geschwistern keine mehrfachen Elementtypen existieren. Im Beispiel existieren nur die Elementtypen Person und Article. Die Selektivität bestimmt sich somit zu 1/2. Der Wert für die Kindsknoten von Article betrage 1/3. Die Selektivität *sel* eines Pfades *path* ergibt sich nunmehr aus dem Gesamtprodukt der Werte der lokalen Selektivität eines Elementtyps e_i mit der Länge n:

$$sel(path) = \prod_{i=1}^{n} s(e_i)$$

Somit ergibt sich für path=/uv/article/year: sel(/uv/article/year) = 1 * 1/2 * 1/3 = 1/6. Diese Selektivität für Pfade wird im Falle der Betrachtung der Selection-Operatoren durch Multiplikation mit einem Standardwert für die Selektivität des Prädikats zu einer Selektivitätsabschätzung für einen konkreten XPath-Ausdruck erweitert. Für Punktanfragen (eq) wird ein Standardwert von 1/10, für Bereichsanfragen (lt, le, gt, ge) 1/3 und für die Überprüfung der Ungleichheit (ne) 1 - 1/10 = 9/10 angenommen. Die Selektivität kann nun benutzt werden, um die Größe der jeweiligen Zwischenergebnisse zu bestimmen. Allerdings gilt, dass je spezieller ein Ausdruck wird, zwar die Zwischenergebnisse entsprechend minimiert werden, aber die gemeinsame Verwendung dieser Ergebnisse bei parallelen Anfragen eingeschränkt ist. Solche Ergebnisse lassen sich nicht sinnvoll in einem Cache vorhalten, da ihre Wiederverwendung als nicht sehr wahrscheinlich gilt. Daher wurden Schwellwerte eingeführt, die eine minimale Grenze und eine maximale Grenze definieren, ab der ein Dokument immer, und zwar unabhängig von der Größe des zu erwartenden Zwischenergebnisses, vollständig geholt wird bzw. ab der ein Dokument trotz einer zu geringen Selektivität in keinem Fall mehr vollständig geholt wird. Werden Dokumente vollständig geholt, lassen sie sich natürlich optimal auch bei der Beantwortung anderer Anfragen nutzen. Ob in diesem Zusammenhang Dokumente vollständig geholt werden oder nur Fragmente, entscheidet dynamisch die Ausführungseinheit bzw. die Unified-View-Komponente, die logisch der Realisierung des Fusion-Operators entspricht.

Materialisierte Sichten (Caching)

Es ist bei der Anfrageverarbeitung durchaus vorstellbar, dass die entstandenen Zwischenergebnisse über die Dauer genau einer Anfrage hinausgehend vorgehalten werden (Caching) oder Anfrageer-

gebnisse aus vorgenerierten XML-Fragmenten beantwortet werden. Entsprechende Direktive können bereits bei der Registrierung angegeben werden. Die Verwendung vorgenerierter XML-Fragmente entspricht einem Vorladen der Cache-Komponente.

3.11.7 Beispiel einer optimierten Anfrage

Die zuvor beschriebenen Prinzipien werden nun zur Vervollständigung des Anfrageplans herangezogen. Die Access-Operatoren werden hinzugefügt, die Projektion abgeleitet und die entsprechenden Selektionen in den Anfrageplan eingefügt. Dies ist in Abb. 3.25 dargestellt.

Translation: Erstellung eines Ausführungsplans

Nach der Optimierung ist der Zugriff auf die Datenquellen zwar vollständig vorgeplant und könnte zumindest logisch bereits ausgeführt werden. Zuvor muss allerdings der Anfrageplan um konkrete Informationen über die zu verwendende Implementierungen unter Berücksichtigung der Möglichkeiten und Fähigkeiten der Datenquellen erweitert werden. Bei der Translation in SHARX geht es in erster Linie aber darum, zunächst überhaupt einen passenden Ausführungsplan zu finden[18].

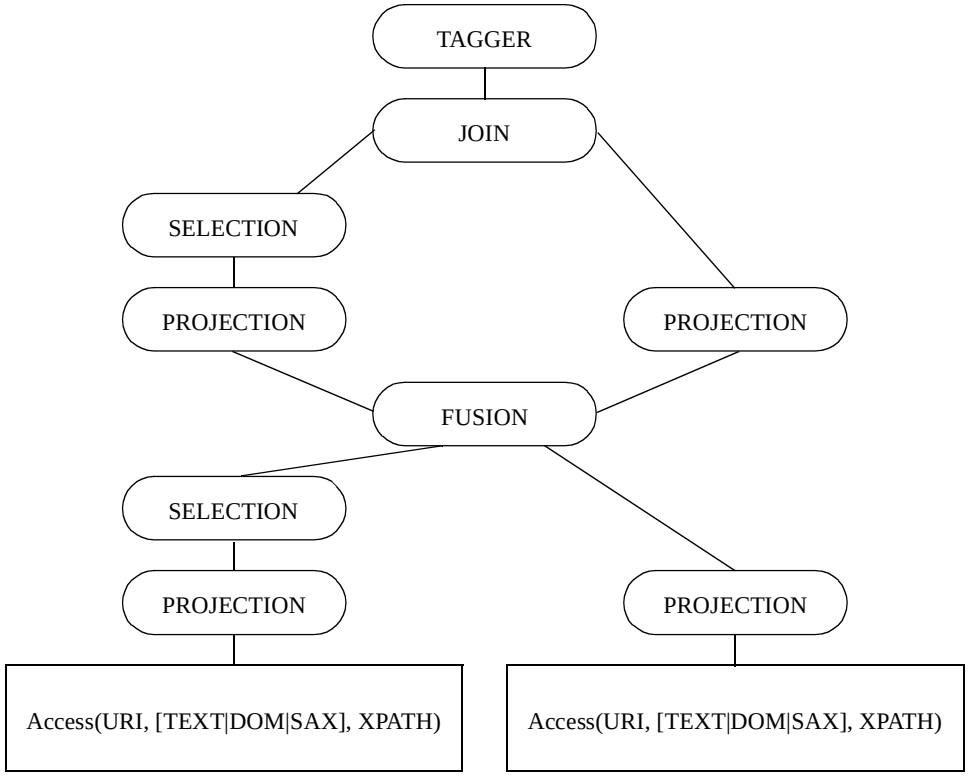


Abb. 3.25: Optimierter Beispielanfrageplan

Der Zugriff auf die Datenquellen geschieht logisch mittels des Access-Operators, der durch die Data-Provider-Komponente realisiert ist. Die verwendete XML:DB-Schnittstelle bietet bereits die Funktionalität über verschiedene XML-Schnittstellen (DOM, SAX, Text) auf die Datenquelle zuzugreifen. Auch die Verwendung von XPath ist als ein Dienst (Service) in die XML:DB-Schnittstellenspezifikation integriert. Projection- und Selection-Operatoren bedienen sich zweier generi-

18.) Eine Optimierung wäre natürlich sehr hilfreich, aber würde auch ein Kostenmodell erfordern. Diese Art der Optimierung wird vorerst nicht betrachtet und ist späteren Arbeiten vorbehalten.

scher XPath-Implementierungen für SAX (z. B. XSQ[19]) [PC03a, PC03b] und DOM (z. B. Jaxen[20]). Insbesondere können die verschiedenen Projektionspfade auch strombasiert ohne mehrfaches Verarbeiten desselben XML-Datenstroms ausgewertet werden [LCH+02]. Der Fusion-Operator ist durch die Unified-View-Komponente realisiert.

Die Translation folgt einer Präferenzliste und bevorzugt momentan die Verwendung von DOM. Es wird daher versucht, an alle Operatoren eine DOM-Implementierung zu binden. Gelingt dies nicht, folgt die Verwendung von SAX und dann erst Text. Es werden allerdings in den Fällen, in denen keine DOM-Implementierung verfügbar ist, Convert-Operatoren an den entsprechenden Stellen im Ausführungsplan eingefügt, um der Data-Provider-Komponente einen Hinweis auf den zu verwendenden Adaptor zu geben. Gemäß der Fähigkeiten der Datenquellen wird versucht, die Projektion und Selektion in den Datenquellenzugriff zu integrieren. Ist die Datenquelle nicht in der Lage, XPath-Ausdrücke der Version 2.0 zu verarbeiten, sollte die Projektion SAX-basiert SHARX-seitig ausgeführt werden.

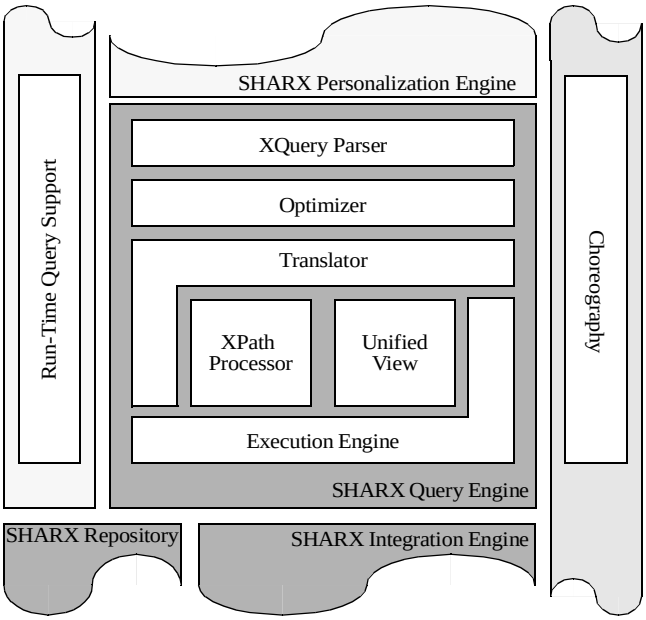


Abb. 3.26: SHARX Query Engine

3.12 SHARX Query Engine

In Abb. 3.26 ist die Komponentensicht der Anfrageverarbeitung dargestellt. Die Anfragekomponente befindet sich oberhalb der Integrationskomponente (Integration Engine) und realisiert die bisher vorgestellte Funktionalität. Die Anfrage wird analysiert, optimiert, transformiert und ausgeführt. Zentrale Komponenten bei der Ausführung sind die XPath-Processor- und Unified-View-Komponente. Die Anfrageverarbeitung zielt hauptsächlich darauf ab, eine XQuery-Anfrage auf XPath abzubilden und einen möglichst großen Teil der Anfrage an die Datenquellen zu delegieren. Die Aufgaben und Konzepte der jeweiligen Komponenten werden im Folgenden erläutert.

19.) http://www.cs.umd.edu/projects/xsq/
20.) http://jaxen.org/

3.12.1 XQuery Parser

Die XQuery-Parser-Komponente ist für die Erstellung der Interndarstellung verantwortlich. Diese entspricht einem Anfragegraphen wie in Abb. 3.24. Dies erfordert zunächst das Übersetzen (Parsing) der Anfrage, um zu einem abstrakten Syntaxbaum zu gelangen. Hierbei wird bereits die Anfrage auf syntaktische Korrektheit hin überprüft. Die Überprüfung auf semantische Korrektheit und die Normalisierung gehen der Generierung des nicht optimierten Anfrageplans voraus. Während der Generierung gehen grundsätzliche Heuristiken der Optimierung der XQuery-Anfrage bereits ein, d. h., Prädikate werden entsprechend aufgespalten und möglichst so in den Anfrageplan integriert, dass sie nach Möglichkeit sehr früh ausgeführt werden können (Selection Push Down), um die während der Anfrageausführung zu verarbeitenden Daten zu minimieren. N-äre Join-Operatoren werden in binäre Operatoren aufgespalten und die für den Join notwendigen Prädikate werden mit dem Join-Operator kombiniert. Die Möglichkeit der Konstantenpropagierung wird ebenso betrachtet, d. h., es wird bei der Erstellung des nicht optimierten Anfrageplans untersucht, ob einfache Prädikate bei geltender Transitivität (Ist A = B und A = $const_i$, dann ist auch B = $const_i$)[21], nicht nur an einen Zweig gebunden, sondern auf andere Pfade übertragen werden können. Die Möglichkeiten sind allerdings sehr gering, da die besondere Semantik der XQuery-Funktionen und Operanden Umformungen mittels der bekannten Idempotenzregeln, also die Anwendung von Äquivalenzumformungen der Boolschen Algebra, nur in Sonderfällen zulassen.

3.12.2 Optimizer

Während der Optimierung wird der Anfrageplan im Hinblick auf den Zugriff auf die Datenquellen optimiert. Zu diesem Zweck wird zuerst die Menge der in Frage kommenden Datenquellen bestimmt und im Anfrageplan berücksichtigt. Nachfolgend werden die vorhandenen Selection-Operatoren, die oberhalb des Fusion-Operators stehen, analysiert, transformiert und evtl. eingefügt. Die Einfügung kann abhängig gemacht werden von der Selektivität eines Ausdruckes unter Berücksichtigung der Größe eines Dokuments. Ist ein Dokument nicht größer als ein vorgegebener Schwellwert für die minimale Größe, so wird das ganze Dokument geholt. Ist die Selektivität zu gering und die Zwischenergebnisgröße erreicht nicht den Schwellwert für die maximale Größe, dann wird das Dokument ebenfalls vollständig angefragt. Gleiches gilt für die Projektion. Die Projektionsvorschrift ergibt sich beispielsweise durch Anwendung von [MS03] und kann zusammen mit der Selektion als ein kombinierter XPath-Ausdruck der Datenquelle zur Auswertung übergeben werden. Ergebnis der Optimierung ist ein Anfrageplan, der auch unterhalb des Fusion-Operators nun sinnvolle Anweisungen enthält.

3.12.3 Translator

Der vollständige Anfrageplan wird durch den Translator in einen Ausführungsplan übersetzt. Dabei werden unter Berücksichtigung der Fähigkeiten der Datenquellen und der Eigenschaften der Dokumente die Zugriffsmethode (Access-Method) festgelegt und evtl. Convert-Operatoren eingefügt. Diese Festlegung auf ein Format (DOM, SAX, Text) geschieht immer auf dem Hintergrund der verfügbaren Operatorimplementierungen. Jedem Operator im Graphen wird in einem weiteren

21.) Im Allgemeinen sind die Vergleichsopatoren in XQuery aber weder transitiv noch reflexiv!

Schritt eine entsprechende Implementierung zugewiesen. Dabei sollte auch die Größe der zu verarbeitenden Daten berücksichtigt werden. Eine entsprechende Metrik für die Beschreibung der Größe könnte auf die Kardinalität der Menge aller Knoten in allen Items zurückgeführt werden, die durch eine entsprechende Abschätzung auf der Grundlage von Statistiken oder der Selektivität in Bezug zur Gesamtgröße der jeweiligen Dokumente gewonnen werden könnte. Das Ergebnis des Translationsvorganges ist ein annotierter Anfrageplan, der um die konkreten Ausführungsdirektive ergänzt worden ist und ausgeführt werden kann. Diese Form des Anfrageplans wird im Folgenden Ausführungsplan genannt.

3.12.4 Execution Engine

Ein Ausführungsplan wird in der Execution Engine ausgeführt. Hierzu wird zunächst der untere Teil des Ausführungsplan analysiert und für den Fusion-Operator die entsprechende Instanz der zugehörigen Implementierung gebildet. Direktive zum Zugriff auf die Datenquellen werden entsprechend vermerkt. Für das Beispiel der in [Tuc04] vorgestellten DOM-Implementierung bedeutet dies konkret, dass entsprechend des globalen Schemas die strukturierenden Elemente und die Elemente, die aus den Dokumenten stammen und keine Daten als Inhalt besitzen, eine DOM-basierte Struktur bilden. Durch Analyse der Access-, Projection- und Selection-Operatoren werden in dieser Struktur an entsprechenden Stellen Direktiven zum Zugriff auf die Datenquellen vermerkt, die bei dem ersten Zugriff auf diesen Knoten ausgeführt werden, d. h., dass die Datenquellen angefragt werden. Das Anfrageergebnis kann materialisiert werden. Dies bietet sich an, wenn beispielsweise die Daten als Text repräsentiert vorliegen. Die XML:DB-API-verwendenden Data Provider erlauben aber auch selbst den Zugriff auf das von ihnen gelieferte Anfrageergebnis über eine DOM-Schnittstelle, so dass die DOM-basierte Fusion-Operatorimplementierung Aufrufe auch delegieren kann und die Daten nicht materialisieren muss.

3.12.5 XPath Processor und Unified View

Die Ausführungseinheit (Execution Engine) bedient sich zweier zentraler Komponenten: XPath Processor und Unified View. Eine XQuery-Anfrage wird durch die in den vorherigen Abschnitten beschriebenen Komponenten analysiert, zerlegt und ausgeführt. Die Grundidee der verfolgten Strategie bei der Anfrageverarbeitung ist es, die XQuery-Anfrage weitestgehend auf XPath abzubilden und die übrige Funktionalität in SHARX-spezifischen Query-Komponenten zu realisieren. Dies erlaubt den Einsatz generischer XPath-Implementierungen. Die SHARX-Komponente XPath Processor in Abb. 3.26 verwendet jeweils eine generische XPath-Implementierung für SAX-basierte und DOM-basierte Verarbeitung. Die Implementierungen des Selection- und Projection-Operators bedienen sich dieser Komponente. Die zweite zentrale Komponente bei der Ausführung ist die Unified-View-Komponente. Sie entspricht der Realisierung des Fusion-Operators und kann ebenso wie die XPath-Processor-Komponente in zwei Implementierungen (SAX und DOM) vorliegen.

Caching

Im Falle einer DOM-basierten Verarbeitung ist es möglich, benötigte Daten erst bei Bedarf anzufragen (Navigation-driven Evaluation, siehe auch [PV02]) und Zwischenergebnisse für weitere Anfragen in der Unified-View-Komponente zwischenzuspeichern. Sie damit also in einem Cache zu

puffern. Die navigationsgetriebene, puffernde Verarbeitung erfordert bei der Beantwortung mehrerer paralleler Anfragen auf einem Unified View Cache, zumindest eine einfache Synchronisation zur Steuerung konkurrierender Zugriffe, die sogar im reinen Nur-Lese-Betrieb auftreten. Durch die Cache-Funktionalität der Unified-View-Komponente werden nicht nur Zwischenergebnisse gepuffert, sondern es müssen auch Daten verdrängt werden können. Um feststellen zu können, auf welche Daten nicht mehr zugegriffen wird, wird im Falle der DOM-basierten Unified-View-Implementierung ein leichgewichtiges Sperrprotokoll eingesetzt, das allein mit einfachen RX-Sperren auskommt. Auch in Stream-basierten Ansätzen ist die Verwendung von Caching vorstellbar [DFK+04].

3.12.6 Integration von Beziehungen

Konnten bereits einige der Anforderungen zur Integration und Unterstützung von Beziehungen während der Behandlung der statischen Integrationsaspekte erfüllt werden, so können weitere Anforderungen innerhalb der Unified-View-Komponente realisiert werden. Die physische Unterstützung von Beziehungen auf der Ebene des Unified View zielt auf die Unterstützung globaler Beziehungen ab. Benutzerspezifische, lokale Beziehungen werden in der Personalisierungskomponente durch die Umformulierung von Anfrage und entsprechender Transformation des Anfrageergebnisses realisiert. Die Behandlung globaler Beziehungen ist nach der Definition für alle Benutzer gleich und umfasst folgende Aufgaben [Tuc04]:

- Sicherstellen der Eindeutigkeit wertbasierter Beziehungen (IDs bzw. KEYs),
- Anpassung der ursprünglichen Pfade im Unified View nach der Integration,
- Transformation der zur Darstellung einer Referenz verwendeten XML-Technik in die als Repräsentationsform spezifizierte XML-Technik,
- Einbettung von Daten bei mittels XInclude repräsentierter Beziehungen.

Die schemarelevanten Teile der Unterstützung sind bereits bei der XML-Integration geschehen: also die Berücksichtigung der evtl. notwendigen Transformation bei der Wahl einer entsprechenden Repräsentationsform durch Hinzufügen oder Ändern der betroffenden Attribute und Elemente. Insbesondere bei XInclude sind weitreichende Schemaänderungen notwendig, da ganze XML-Fragmente an entsprechenden Stellen im Unified View ergänzt werden müssen.

3.12.7 Run-Time Query Support

Die gesamte Anfrageverarbeitung und auch die Personalisierung werden durch die gemeinsame Komponente Run-Time Query Support unterstützt. Sie erlaubt den Zugriff auf die Spezifikationen von Beziehungen. Ferner können entsprechende Repräsentationen (DTD und XML-Schema) für die internen Schemata unter Berücksichtigung von globalen Beziehungen zur Verfügung gestellt werden. Hiermit ist die Grundlage für ein umfassendes Personlisierungskonzept geschaffen. Auf der Basis des Unified View können komplexe Anfragen gestellt werden, um unterschiedlichste externe Sichten zu spezifizieren. Die Ergebnisse dieser Anfragen sind mittels XML repräsentiert und können durch standardisierte Transformationsvorgaben (XSLT) in verschiedene endgerätespezifische Präsentationsformen gebracht werden.

4 Inhaltliche Dokumentenverwaltung in XCoP

4.1 Anforderungen an die Dokumentenverwaltung

Für ein solches weitreichendes Personalisierungskonzept ist die Trennung von Inhalt, Struktur bzw. Logik und Präsentation existentiell. Lehrinhalte sollten zum einen nicht mit Präsentationsinformationen überladen und zum anderen durch explizite Beschreibung der Struktur zu einem virtuellen Kursangebot aggregiert werden können. Aber erst diese explizite Beschreibung erlaubt ein adaptives Systemverhalten hinsichtlich des Wissensstandes, der Auffassungsgabe oder des Lerntypus eines Lernenden. Sie kann auch als Maßstab für den Lernfortschritt dienen und erlaubt das einfache Fortsetzen der Lernaktivitäten an der gleichen Stelle zu einer anderen Zeit. Wiederverwendung von Elementen einer virtuellen Lernumgebung ist auf verschiedenen Ebenen denkbar. Es können Lerninhalte, strukturierte Aggregationen von Lerninhalten oder ganze Kurse, inklusive ihrer Präsentation, wiederverwendet werden. Standardisierung ist hierbei ein sehr gewichtiger Punkt (XML, Instructional Management System Project[22] oder Synchronized Multimedia Integration Language[23] (SMIL)).

In der herkömmlichen, auf XML basierenden Standardtechnologie [BPS98] enthalten Dokumente textuelle Daten (als so genannten Content oder Dokumenteninhalt) kombiniert mit strukturellen Informationen, welche die Struktur der textuellen Daten näher beschreiben. Um nun Dokumente aus ihren inhaltlichen Bestandteilen (Fragmenten) flexibler aufbauen und in unterschiedlichen Zusammensetzungen den Benutzern anbieten zu können, ist es erforderlich, die Strukturinformation von den Dokumenteninhalten zu trennen und somit die Beziehungen zwischen den einzelnen Dokumentfragmenten explizit zu verwalten. Eine explizite und separierte Nutzung der Strukturinformation verbessert die inhaltliche Verarbeitung von XML-Dokumenten auf vielfältige Weise. So ist eine Wiederbenutzung von Dokumentenfragmenten möglich, um Datenredundanz und als Folge davon Änderungsanomalien bei replizierten Daten zu vermeiden. Weiterhin ermöglicht eine solche Vorgehensweise die kooperative Bearbeitung von Dokumenten, indem auf Fragmentebene Synchronisation und Versionierung unterstützt werden. Dabei kann die Fragmentgranularität flexibel konfigurierbar gestaltet werden.

Unser Ansatz zur Dokumentenverwaltung greift diese Idee der Trennung von Inhalt und Struktur auf. Dabei soll die zugehörige Systemlösung folgende Anforderungen erfüllen:

- Selektive Wiederbenutzung von Dokumentinhalten
- Ausnutzung von Strukturinformation, z.B. zur Verbesserung der Suchqualität und der Konsistenzprüfung
- Konsistenzkontrolle von Inhalten
- Integrierte Verwaltung von XML-Dokumenten und anderen Ressourcen
- Nutzung von neuer DB-Technologie
- Dynamische Generierung von Dokumenten
- Bereitstellung weiterer Repository-Funktionalität.

22.) http://www.imsproject.org/
23.) http://www.w3c.org/smil/

Für diese Anforderungen entwickelten wir eine Reihe neuer Konzepte, um Dokumentfragmente DB-gestützt [SBM98] modellieren, verwalten und verarbeiten zu können. Diese Konzeptfindung und die zugehörige Systemlösung fassen wir unter dem Akronym XCoP (XML content repository) zusammen.

4.2 Das Conten-Management System XCoP

Wir wollen hier die wesentlichen Aspekte der Verwaltung von Dokumenteninhalten, dem so genannten Content-Management, skizzieren. Dazu müssen die textuellen Repräsentation von XML-Dokumenteninhalten, textuelle Inhalte genannt, und die zugehörige Strukturinformation näher betrachtet werden. Zu den textuellen Inhalten gehören DTDs, Dokumente und (logische) Dokumententeile. XCoP verwaltet sowohl die textuellen Inhalte als auch die zugehörige Strukturinformation der XML-Dokumente. Zu diesem Zweck muss für diese Objekte ein konzeptionelles Modell und eine Menge von Operationen bereitgestellt werden. Auf der konzeptionellen Ebene ist alles, was durch XCoP verwaltet wird, als Repository-Objekt modelliert [BD94]. Solche Repository-Objekte besitzen eine Reihe von Attributen (z.B. Besitzer, Erstellungsdatum, letzte Modifikation usw.) und gliedern sich in verschiedene Typen wie Ressourcen, Dokumente, Fragmente, Elemente usw.

4.2.1 Verwaltung textueller Inhalte

Die herkömmliche XML-Dokumentenverwaltung besitzt keine Funktionen zur Wartung und Konsistenzsicherung von zusammengehörigen Dokumententeilen. Änderungen in der physischen Struktur eines Dokumentes erfordert (ggf.) manuelles Nachziehen dieser Änderungen in der DTD. Im Gegensatz dazu gestattet XCoP den Benutzern die Dokumenteninhalte logisch aufzuteilen, ohne die DTDs akutalisieren zu müssen; die Wartung und Konsistenzsicherung der einzelnen Teile erfolgt automatisch. Dazu wird ein geeignetes Fragmentkonzept bereitgestellt.

Definition von Fragmenten

Ein Fragment [GV99] besteht aus textuellen Inhalten und kann weiterhin eine Menge anderer Fragmente (Kindfragmente) umfassen, aber nicht sich selbst. Die Eigenschaften von XML-Dokumenten wie Wohlgeformtheit und Validität sind entsprechend definiert. Wie in Abb. 4.1 veranschaulicht, können wir basierend auf dem Fragmentkonzept XML-Dokumenteninhalte sowie ihre DTDs in flexibler Weise in Teile zerlegen, die zum Aufbau neuer (DTD-) Fragmente wiederbenutzt werden können. Abb. 4.1 illustriert ein Beispiel für die Fragmentwiederbenutzung und die Strukturbeziehungen, die einen so genannten Fragmentgraph aufspannen (der die Aggregationsbeziehungen zwischen Vater- und Kindfragmenten beschreibt).

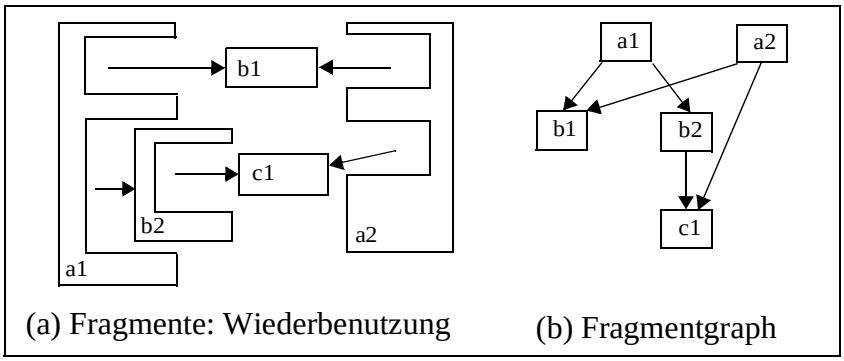


Abb. 4.1: Beispiel für Fragment-Reuse und sein Fragmentgraph

Konzeptionelles Modell für die Verwaltung textueller Inhalte

Die konzeptionelle Modellierung von textuellen Inhalten ist in Abb. 4.2 gezeigt. Sie umfassen XML-Dokumente, DTDs, Fragmente und andere Arten von Ressourcen. In unserem Modell ist eine Ressource durch ein Repository-Objekt repräsentiert, was durch ein URI identifiziert werden kann. Ressourcen können entweder Binärressourcen (z.B. Bilder oder ausführbare Dateien) oder textuelle Ressourcen (z.B. ASCII-Textdateien) sein. Diese werden wiederum verfeinert in DTDs, Dokumente und Fragmente. Ein Fragment kann aus mehreren Kindfragmenten bestehen, die geordnet sind und gemeinsam von mehreren Vaterfragmenten benutzt werden können. Fragmente enthalten Markup-Elemente und lassen sich zu DTD-Fragmenten spezialisieren, die aus Markup-Deklarationen bestehen.

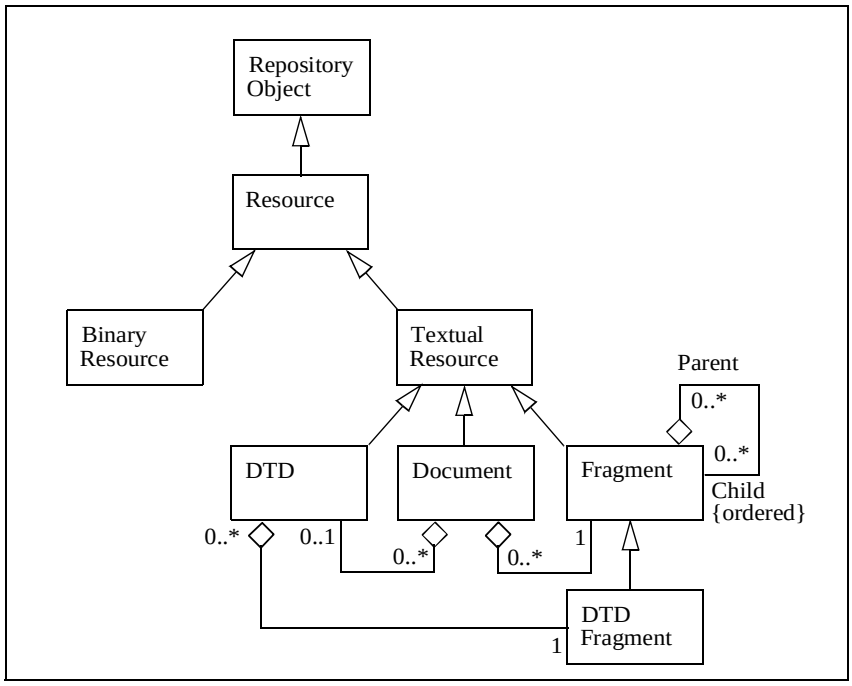


Abb. 4.2: Konzeptionelles Modell zur Verwaltung textueller Inhalte

4.2.2 Ausnutzung struktureller Information

Es werden verschiedene Arten von struktureller Information genutzt, um die automatische Konsistenzsicherung von Dokumentinhalten zu unterstützten und um erweiterte Suchmöglichkeiten bereitzustellen.

- **Information aus wohlgeformten XML-Dokumenten**
  Dazu gehören Markup-Elemente, Attribute, Referenzen zwischen Markup-Elementen, Kompositionsbeziehungen verschiedener Art sowie die Ordnung von Markup-Elementen
- **DTD-Information**
  Diese umfasst Entity-Deklarationen, Element-Deklarationen, Attributlistendeklarationen sowie verschiedene Arten von Kompositionsbeziehungen
- **XLink-Information**
  Hierzu gibt es verschiedene Typen von Link-Elementen zusammen mit XLink-spezifischen Attributen wie type, href, role, arc usw.

Konzeptionelles Modell zur Verwaltung struktureller Information

Abb. 4.3 stellt das konzeptionelle Modell für die Verwaltung von strukturellen Informationen dar. Danach hat ein Fragment eine geordnete, nicht-leere Menge von Markup-Elementen. Jedes von diesen kann ein oder mehrere Attribute besitzen. Wir unterteilen die Attribute weiterhin in die Typen ID und IDREF(s). Danach kann ein Markup-Element mit ID-Attributen referenziert werden durch ein oder mehrere andere Elemente via IDREF(S)-Attribute. Andererseits verweist ein Element mit einem IDREF(S)-Attribut genau auf ein bzw. mehrere Elemente mit jeweils einem ID-Attribut. In Abb. 5 ist weiterhin ein vereinfachtes konzeptionelles Modell von XLink dargestellt. Die verschiedenen Arten von Link-Elementen in XLink sind als Typen von Markup-Elementen modelliert. Ein erweiterter Link hat eine nicht-leere, geordnete Menge von Locator-Elementen. Schließlich verweist ein einfacher Link oder ein Locator-Element auf genau eine Ressource.

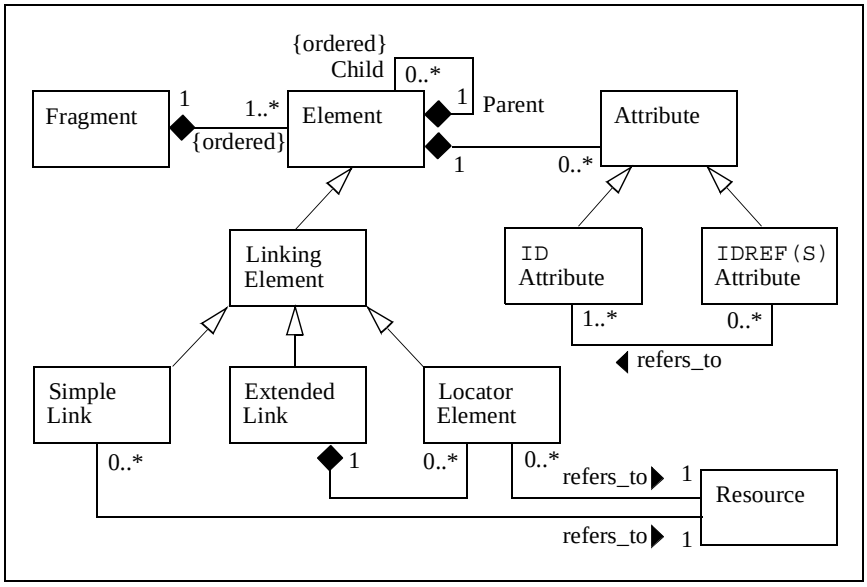


Abb. 4.3: Modellierung structureller Information in Fragmenten

Auf weitere Detaillierungen der Modellierung (z.B. von DTD-Fragmenten) wollen wir in diesem Überblick nicht eingehen (siehe [SRL00]).

4.2.3 Verarbeitung von Repository-Objekten

Für die Dokumentenverwaltung sind geeignete Funktionen bereitzustellen, die sich natürlich vor allem an unserem Fragmentkonzept und den strukturellen Beziehungen auszurichten haben. Wir unterscheiden dabei die folgenden vier Kategorien:

- **Aktualisierungsoperationen auf Repository-Objekten**
  Es sind Operationen zum Erzeugen, Ändern und Löschen bereitzustellen, wobei jeweils die durch die DTD vorgegebenen Konsistenzanforderungen zu beachten sind.
- **Manipulationsoperationen auf Beziehungen**
  Diese Operationen werden eingesetzt, um explizit Beziehungen zwischen Repository-Objekten einzurichten und aufzugeben.
- **Retrieval-Operationen**
  Sie erlauben den Zugriff auf textuelle Inhalte von einem Repository-Objekt und stellen eine Navigationsmöglichkeit auf Beziehungen zwischen Repository-Objekten bereit.
- **Spezielle Operationen**
  Diese Operationen sind exklusiv anwendbar auf bestimmte Repository-Objekttypen. So

kann beispielsweise von einem Fragment-Objekt ein Teil abgespalten werden als neues Kindfragment (splitFragment). Weitere wichtige Operationen sind Zusammenfassung von Fragmenten (mergeFragment) oder die Überprüfung von Inhalten (matchFragment).

4.3 Implementierung

Zur Realisierung von XCoP ziehen wir als Infrastruktur unser iWebDB, ein integriertes Framework für Web-Datenbanken [Loe01a, LR99], heran. Bevor wir spezielle Aspekte der Dokumentenverwaltung beleuchten, geben wir einen kurzen Abriss über die iWebDB-Architektur und -Funktionalität:

4.3.1 iWebDB

iWebDB basiert auf einem objekt-relationalen Datenbanksystem (ORDBS) und nutzt OR-Erweiterbarkeitsmechanismen wie UDT, UDR und Zugriff auf externe Daten [SBM98]. Wie in Abb. 4.4 veranschaulicht, besteht iWebDB momentan aus 8 Modulen, wovon 5 Modulen DBS-Erweiterungen, so genannte DataBlades oder Extenders sind: Doc, ED (External Data), eXtract, DG (Document Generator) und XCoM (XML Content Manager). Zusätzlich gibt es zwei Client-Applikationen, SM (Site Manager) und XCoEx (XML Content Explorer). Der Modul MP (Macro Processor) ist eine Funktionsbibliothek, die für Client-seitige als auch für Server-seitige Applikationen genutzt werden kann [Loe01a]. XCoM und XCoEx sind spezifische Komponenten von XCoP.

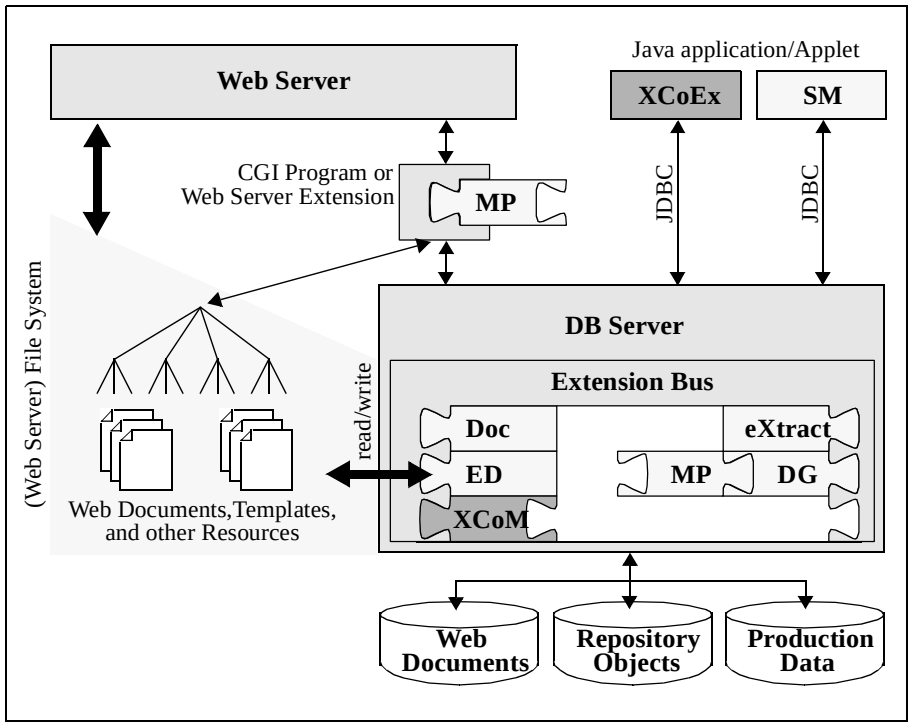


Abb. 4.4: Architektur von iWebDB

iWebDB/Doc

Dieser Modul bildet die Grundlage für die datenbankinterne Dokumentenverwaltung. Der von ihr zur Verfügung gestellte Datentyp Content dient in seinen einzelnen Spezialisierungen der Aufnahme verschiedener Dokumenttypen und enthält neben dem eigentlichen Dokumentinhalt einige für den jeweiligen Typ (XML, HTML, Grafik, Sound) charakteristische Attribute, wie z. B. Titel, Hyperlinks oder auch Formatangaben. Neben dem Speichern bietet er Funktionalität zur Verwaltung, Anfrage und Analyse von Inhalten.

iWebDB/ED (External Data)

Er nutzt die von iWebDB/Doc zur Verfügung gestellten Datentypen Document und Content, um den Zugriff auf extern gespeicherte Dokumente über die Protokolle file:, ftp: und http: zu ermöglichen. Auf diese Weise kann sowohl mit Hilfe von dateibasierten Werkzeugen als auch mittels SQL-Anfragen auf externen Dokumenten gearbeitet werden. iWebDB/ED wird vom Dokumentengenerator genutzt, um generierte Dokumente ins Dateisystem zu schreiben oder mit Hilfe von Dokumentschablonen aus dem Dateisystem zu lesen.

iWebDB/eXtract

Um die Suchfunktionalität von iWebDB zu verbessern, stellt er zusätzliche Funktionalität zur Analyse von Dokumenten bereit. Zu Ablage von extrahierten Daten werden benutzerdefinierte Indexstrukturen verwendet, die von einer Suchmaschine genutzt werden können.

iWebDB/SM (Site Manager)

Dieser Modul stellt eine grafische Benutzerschnittstelle zur Verfügung, mit deren Hilfe die Installation und die Wartung von Web-Dokumenten zentralisiert und vereinfacht wird. Zu seinen Aufgaben gehört auch eine flexible Benutzer- und Gruppenadministration.

iWebDB/MP

Er realisiert einen als Java-Klassenbibliothek bereitgestellten Makroprozessor. Seine Aufgabe ist die Generierung von Web-Seiten anhand spezieller Vorlagedateien, in die Makros eingebettet sind, die vom Makroprozessor erkannt und verarbeitet werden.

iWebDB/DG

Er wird zur DBS-gesteuerten Generierung von Web-Seiten eingesetzt, um so stets veränderliche Dokumente mit aktuellem Inhalt anbieten zu können. Zum Erkennen und Auslösen der Änderungen werden SQL-Trigger eingesetzt.

4.3.2 XCoP

Nach einem Überblick über die iWebDB-Architektur und ihrer wichtigsten Komponenten wollen wir noch kurz zwei XCoP-spezifische Komponenten einführen:

XCoEx

Dieser Modul stellt eine graphische Benutzerschnittstelle zur Manipulation und zum Retrieval von Repository-Objekten zur Verfügung. Zum Editieren von textuellen Inhalten hält er einen Texteditor bereit. Zusätzlich werden Administrationsaufgaben unterstützt, die Repository-Objekte und ihre Abbildung auf das Dateisystem des Web-Servers sowie Zugriffsrechte für Benutzer und Gruppen betreffen. XCoEx ist als Java-Applikation implementiert und benutzt JDBC (Java Database Connectivity).

XCoM

Dieser Modul ist für die Server-seitige Verwaltung von Repository-Objekten verantwortlich. Er implementiert auch das konzeptionelle Modell und die zugehörigen Operationen als Zusatzschicht oberhalb des ORDBS. Zur Charakterisierung der XCoP-Implementierung wird hier beispielhaft die Abbildung des konzeptionellen Modells von XCoP auf ein OR-Datenmodell diskutiert. Die Repository-Objekttypen (aus Abschnitt 4.2) lassen sich direkt auf ein objekt-relationales DB-Schema abbilden (Abb. 4.5). Jeder Objekttyp wird auf einen SQL-Typ (TYPE) und jeder spezialisierte Typ auf einen Subtyp (mit dem Schlüsselwort UNDER) abgebildet. Jeder Subtyp erbt alle Attribute und operationalen Eigenschaften (Routinen, Operatoren, Integritätsbedingungen und Wertebereiche) von seinem Supertyp. Basierend auf der definierten Typhierarchie lässt sich dann entsprechend eine Tabellenhierarchie spezifizieren (Abb. 4.5(a)). Der Primärschlüssel-Constraint, der für die Supertabelle repository_object definiert ist, wird von allen Subtabellen geerbt.

Das verwendete ORDBS (IDS/UDO [Inf97]) bietet ein Überladen von Routinen, das dem Benutzer erlaubt, einen einzigen Namen (z.B. create_object) mehreren Routinen mit verschiedenen Signaturen zuzuweisen (Abb. 4.5(b)). Es unterstützt jedoch keine Referenztypen und gewährleistet nur begrenzte Möglichkeiten, was Mengenoperationen auf Kollektionstypen anbetrifft. Da beide Techniken sehr nützlich für die Implementierung einer erweiterten Beziehungstypsemantik sind, mussten wir sie mit Hilfe von referentiellen Integritätsbedingungen simulieren (Abb. 4.5(c)).

Da ein Fragment die kleinste vom Benutzer spezifizierte Informationseinheit ist und Datenlokalität erwarten lässt, ist es vorteilhaft, es auch als Einheit zu speichern. Fragmente werden deshalb als Zeilen in einer einzigen Tabelle fragment abgelegt. Dieser Speicherungsansatz liefert offensichtlich ein besseres Leistungsverhalten, wenn ein Dokument aus seinen Fragmenten aufzubauen ist, als andere Ansätze, die Dokumente aus ihren Elementen zusammensetzen. Außerdem kann parallele Arbeit auf Fragmenten direkt durch einen fein-granularen Sperrmechanismus kontrolliert werden. Aggregationsbeziehungen zwischen Fragmenten werden über eine separate Tabelle (fragment_graph) abgebildet. Dabei ist die Reihenfolge der Kindfragmente innerhalb eines Vaterfragments explizit zu verwalten (order_pos).

Das Speichern von Strukturinformationen, speziell von Elementen und Attributen, erlaubt viele Variationsmöglichkeiten, ist aber auch leistungskritisch [FK99, STH+99, DFS99]. Basierend auf einem Abbildungsschema aus [FK99] nutzen wir den Attributansatz zum Speichern von Strukturinformationen. Die Abhängigkeiten zwischen einem Fragment und seiner Strukturinformation lassen sich automatisch über einen Trigger kontrollieren. Änderungen in einem Fragment aktivieren einen Trigger, der die zugehörige Strukturinformation wartet (und umgekehrt).

Anfragen in einer XML-Anfragesprache können dann in einfacher Weise in SQL-Anfragen übersetzt werden. Außerdem lässt sich die Suchqualität bei Dokumenteninhalten wesentlich verbessern, wenn ihre Strukturinformation ausgenutzt werden kann [SRL00].

Andere Erweiterungen von iWebDB, insbesondere MP, DG und ED, erlauben eine dynamische Inhaltsgenerierung von Dokumenten und Fragmenten, wobei auch existierende (Altlasten-) Datenbanken (leyacy DBs) und externe Dateien über spezielle Makro-Formatvorlagen integriert werden können. In unserem ersten Prototyp speichern wir textuelle Inhalte als LOBs (large objects), da

```
CREATE TYPE repository_object_t (
   oid VARCHAR(8) NOT NULL, owner VARCHAR(20), cdate DATE);
CREATE TYPE resource_t (
   uri VARCHAR(255), contents BLOB) UNDER repository_object_t;
CREATE TYPE textual_resource_t (
   description VARCHAR(255)) UNDER resource_t;
CREATE TYPE document_t (
   content_status VARCHAR(8), dtd_oid VARCHAR(8), root_fragment_oid VARCHAR(8)
   ) UNDER textual_resource_t;
CREATE TYPE dtd_t (
   name VARCHAR(60), root_fragment_oid VARCHAR(8)) UNDER textual_resource_t;
CREATE TYPE fragment_t (
   is_root BOOLEAN) UNDER textual_resource_t;
CREATE TYPE dtd_fragment_t (
   ) UNDER fragment_t;

CREATE TABLE repository_object OF TYPE repository_object_t (PRIMARY KEY (oid));
CREATE TABLE resource OF TYPE resource_t UNDER repository_object;
CREATE TABLE textual_resource OF TYPE textual_resource_t UNDER resource;
CREATE TABLE document OF TYPE document_t UNDER textual_resource;
CREATE TABLE dtd OF TYPE dtd_t UNDER textual_resource;
CREATE TABLE fragment OF TYPE fragment_t UNDER textual_resource;
CREATE TABLE dtd_fragment OF TYPE dtd_fragment_t UNDER fragment;
```

(a) Definition einer Typhierarchie und einer getypten Tabellenhierarchie

```
CREATE FUNCTION create_object (document) RETURNING VARCHAR(8);
CREATE FUNCTION create_object (dtd) RETURNING VARCHAR(8);
CREATE FUNCTION create_object (fragment) RETURNING VARCHAR(8);
CREATE FUNCTION create_object (dtd_fragment) RETURNING VARCHAR(8);
```

(b) Überladen von Routinen

```
ALTER TABLE document ADD CONSTRAINT (
   FOREIGN KEY (dtd_oid) REFERENCES dtd (oid),
   FOREIGN KEY (root_fragment_oid) REFERENCES fragment (oid));

ALTER TABLE dtd ADD CONSTRAINT (
   FOREIGN KEY (root_fragment_oid) REFERENCES dtd_fragment (oid));

CREATE TABLE fragment_graph (
   parent_oid VARCHAR(8), child_oid (8), order_pos SMALLINT,
   PRIMARY KEY (parent_oid, child_oid),
   FOREIGN KEY (parent_oid) REFERENCES fragment (oid),
   FOREIGN KEY (child_oid) REFERENCES fragment (oid));
```

(c) Implementierung von Beziehungen zwischen Repository-Objekten

Abb. 4.5: Beispiel für ein OR-Schema

IDS/UDO keinen vergleichbaren Mechanismus zu DataLinks [Dav99] anbietet, mit dem Links auf extern verwaltete Dateien eingesetzt und gewartet werden können. Auch hier setzen wir referentielle Integritätsbedingungen und Trigger zur automatischen Kontrolle und Gewährleistung der Link- und Referenzkonsistenz ein.

5 Dokumentenverwaltung und inhaltsbasierte Suche

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 schlüsselwortbasierte Suche und setzen vor allem keine domänenspezifischen 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. Der Prozess der Erschließung lässt sich im Allgemeinen durch den in Abb. 5.1 gezeigten Mehrschrittvorgang beschreiben. Die Abbildung zeigt die Schritte Sammeln/Einstellen, Prüfung (auf Lehrmaterial), Erfassung einfacher Metadaten, bibliothekarische Erschliessung sowie (Qualitäts-)Prüfung. Im Folgenden wollen wir das Problem der Dokumentensuche und Erschließung in einem größeren Zusammenhang sehen und uns dabei auf die Aspekte der Datenverwaltung konzentrieren.

Abb. 5.1: Erschließungsprozess

5.1 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 DTD oder durch ein XML-Schema-Dokument 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.

```
<?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. 5.2: Beispiel einer XML-Repräsentation

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[24] 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. 5.2 aussehen. Wie man in der Abbildung Abb. 5.2 auch leicht sehen kann, werden die Metadatenattribute auf XML-Elemente abgebildet. Angaben über

24.) http://www.dublincore.org

den Ursprung („Origin“) des Inhalts eines XML-Elements bzw. Angaben zum Format oder Schema („Scheme“) werden unter Einsatz von XML-Attributen gemacht. Das mit XML verbundene semistrukturierte Datenmodell erlaubt eine sehr intuitive Verwendung der Metadatenattribute. Bei der Erschließung eines Lehr-/Lernangebots können nicht in jedem Fall alle Metadatenattribute sinnvoll ausgefüllt werden, da es vorkommen kann, dass nicht alle Informationen verfügbar sind. Aber auch Mehrfachnennungen, bei denen nicht von vornherein 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. 5.3. Man erkennt, dass in der Datenbank („Metabase“) Lehr-/Lernangebot („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 zum Beispiel, wie man anhand der DTD sieht, 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.

```
<!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. 5.3: Auszug aus einer DTD

Jedes erfasste Lehr-/Lernangebot wird zunächst durch ein eigenes XML-Dokument repräsentiert. Allerdings können Lehr-/Lernangebote auch in Beziehung („Relation“) zueinander stehen. Die Definition der Metadatenattribute sieht verschiedene Arten von Beziehungen vor: Version, Teil („Part“), Format, sowie Verweise auf Gutachten („Review“) und Benutzerkommentare („User Comment“). Besteht beispielsweise ein Lehr-/Lernangebote aus verschiedenen Kapiteln, so können

```
<?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. 5.4: Beispiel einer Relation.HasPart-Beziehung

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 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-/Lernangebot zu erfassen. Allerdings geht damit die Möglichkeit verloren, zusätzlich einen Autor oder ähnliche Metadatenattribute für die referenzierten

```
<?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. 5.5: Beispiel einer Inline-Definition

Einträge spezifizieren zu können. Aber so wird sehr einfach das Problem umgangen, lokal, d. h. 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 Typ „Learning Resource“ definiert und miteinander verknüpft (siehe: Relation.HasPart.Reference-Element in Abb. 5.4). Müsste man bereits bei der initialen Erschließung Client-seitig die in Abb. 5.4 gezeigte Struktur erzeugen und an die Datenbankkomponente schicken, so ergäbe sich eine Menge vermeidbarer Redundanzen und damit auch eine Menge potenzieller Fehlerquellen. Daher ist für die initiale Erschließung eine Vereinfachung vorgesehen. In Abb. 5.5 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. 5.5 wird an die Datenbankkomponente geschickt und erst dort zu der Struktur aus Abb. 5.4 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. 5.4 zu wählen. Einen entsprechenden Auszug aus der DTD findet sich in Abb. 5.6.

```
<!-- ============= 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. 5.6: Strukturbeschreibung (DTD) der Inline-Definition

5.1.1 Verarbeitungsmodell

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

- ein elementbasiertes und
- ein dokumentbasiertes Verarbeitungsmodell.

Die elementbasierte 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 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 zugrunde liegenden Prozesse und damit vor allem aufgrund des grundlegenden 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[25]) 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.

Dokumentenbasierte 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

25.) http://www.webdav.org

(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 Metadatenattribute auf XML ein konkretes Datenmodell gegeben, das dem Austausch der zu verwaltenden 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.

5.2 WebDAV-basierte prototypische Realisierung mit Slide

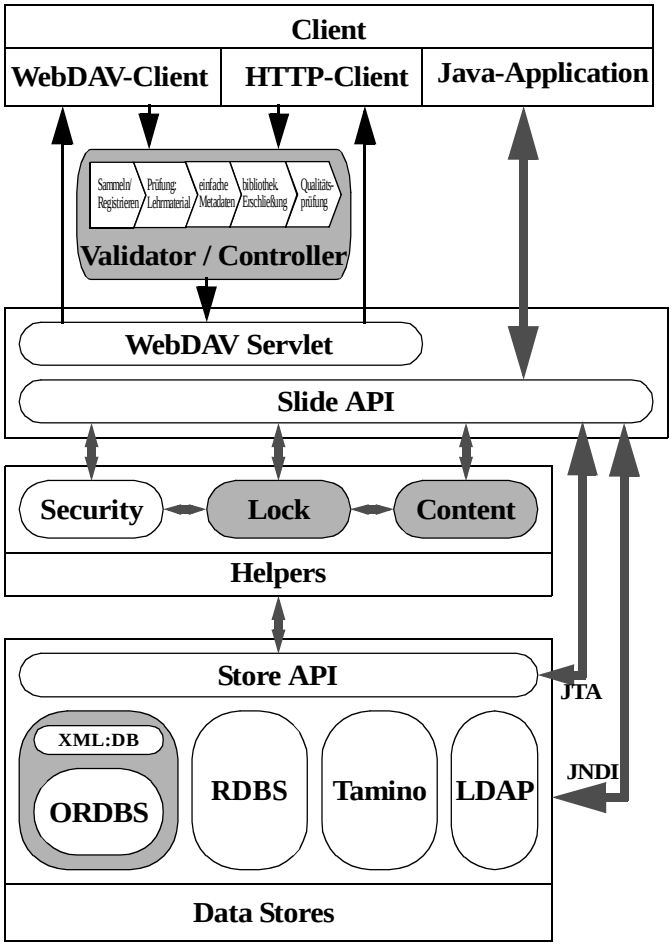


Abb. 5.7: Slide/WebDAV-basierter Architekturentwurf

Eine Realisierung des WebDAV-Standards stellt das Slide-Projekt[26] der Apache-Jakarta-Gruppe[27] 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 zugrunde liegende Systemarchitektur findet man in Abb. 5.7. 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 mögliche 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 Eigenar-

26.)http://jakarta.apache.org/slide
27.)http://jakarta.apache.org

ten Web-basierter Anwendungen. Für die Abbildung der verschiedenen Verarbeitszustände eines Dokuments wurde in naheliegender Weise für jeden Zustand ein eigenes Verzeichnis 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 usw.) können über Properties bezogen werden, so dass die Erschließung optimal unterstützt wird. 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 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 (z. B. Benutzerverwaltung, Sicherungen usw.).

5.3 Produktivsystem

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 zugrunde liegenden Erschließungsprozess und damit auch an die Datenverwaltungskomp-

onente zu einer sehr späten Phase des Projekts nicht unwesentlich geändert hatten. Die erweiterte Implementierung basiert auf dem Java™ 2 Enterprise Edition Framework (J2EE)[28] 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.

5.3.1 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).

5.3.2 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 verschiedene 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.

5.3.3 Enterprise-JavaBeans-Komponenten

Enterprise JavaBeans (EJB) sind Komponenten eines flexiblen Komponentenmodells für Geschäftsobjekte in verteilten Systemumgebungen. Ein Komponentenmodell ermöglicht die Erstellung von wiederverwendbaren Software-Teilen (Komponenten). Es beschreibt eine Infrastruktur für den Einsatz und die Kommunikation von diesen so genannten Komponenten. Eine Komponente

28.) http://java.sun.org/j2ee

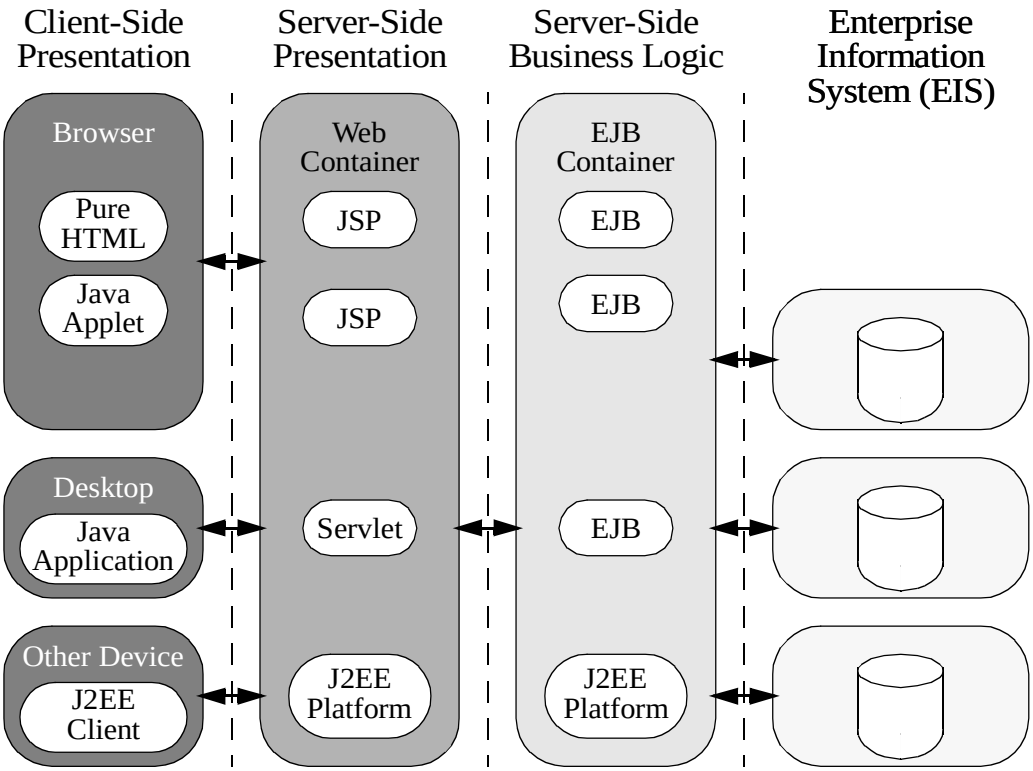


Abb. 5.8: J2EE-Mehrschichtenarchitektur

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).

5.3.4 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 [Fle04]. Die Re-

Implementierung mittels Java 2 Enterprise-Edition (J2EE) umfasst daher folgende Teilbereiche (siehe Abb. 5.9):

- Anfrageverarbeitung (Query Engine)
- Ergebnisverarbeitung (Result Set Processor)
- XML-Dokumenten-Management (XML-Processor)
- Web-basierte Suche (Web Search)
- Redaktionssystem (Indexing Tool)
- Interoperabilität (OAI)
- Support-Komponenten (beispielsweise 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) bearbeitet worden und die Web-basierte Suche und die entsprechende Infrastruktur zum Testen der Such-Schnittstelle waren sehr schnell realisiert bzw. vollständig implementiert [FFW02]. Diese Testumgebung wurde während der Projektlaufzeit sukzessive durch die verschiedenen Projekt- und Diplomarbeiten mit Funktionalität ausgefüllt.

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 Processor“), der Ergebnisverarbeitung (Result Set Processor) und des XML-Dokumenten-Managements („XML-Document 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 Schemainformationen (beispielsweise 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 Volltextsystem (hier: ASP-Seek) unterstützt und bei der Anfrageverarbeitung integriert.

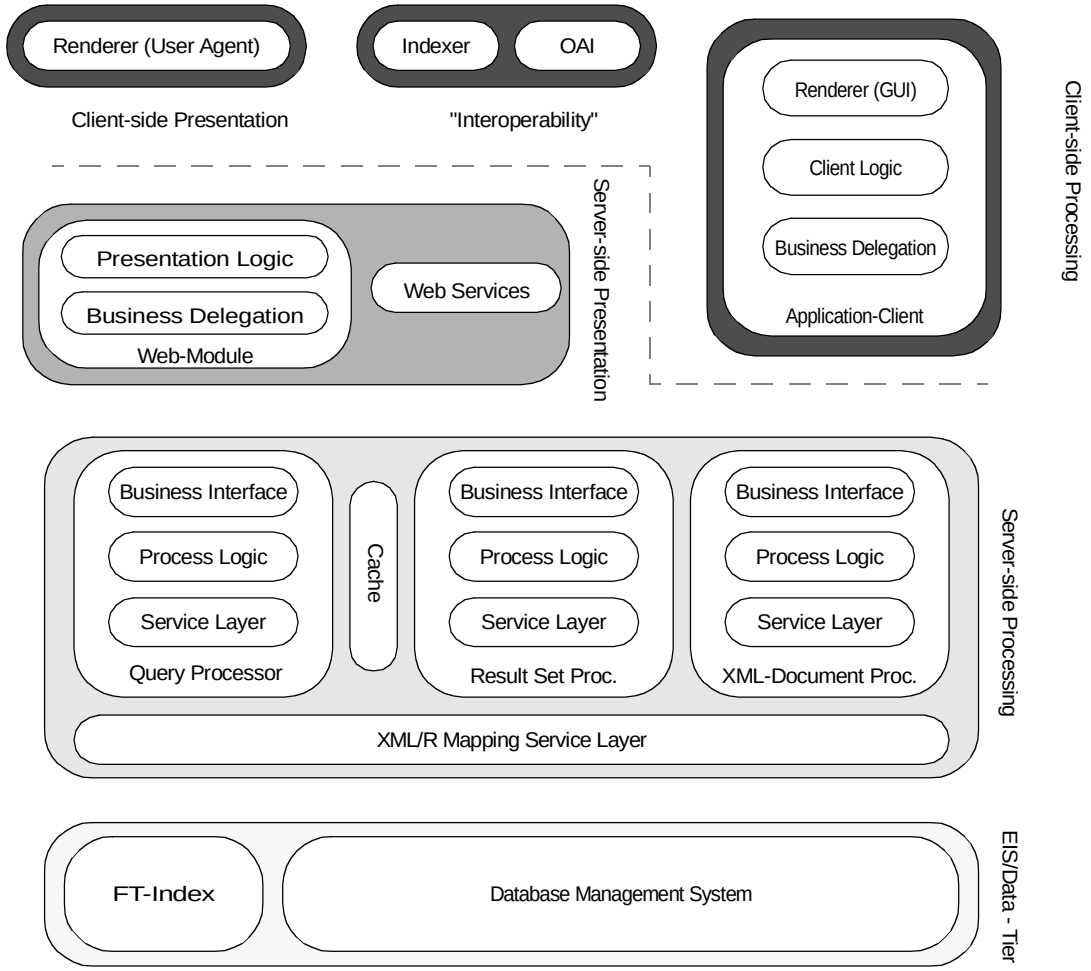


Abb. 5.9: Architekturübersicht

XML-Dokumenten-Management (XML-Document 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 dokumentenorientiert 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 Processor)

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 wiederzuverwenden. Das integrierte Gesamtergebnis wird in der relationalen Datenbank materialisiert und kann von der Ergebnisverarbeitung weiterverwendet werden.

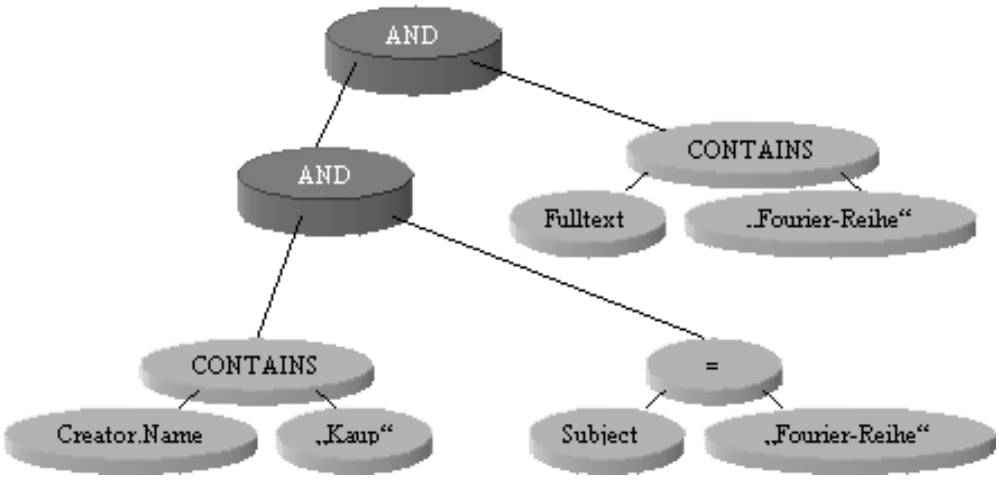


Abb. 5.10: Anfrageverarbeitung mittels eines Query Tree

Die Anfrageverarbeitung führt somit die Anfrage nur in dem Sinne aus, dass sie die interne Repräsentation einer Anfrage („Query Tree“, Abb. 5.10) 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 datenbankspezifische Cursors zum Durchlaufen der Ergebnismenge. Dies schont zwar die datenbankseitig aufzubringenden Ressourcen für die Speicherung der Ergebnisse, allerdings erhöht sich die Anzahl der offenen Datenbankverbindungen, die nicht gemeinsam benutzt werden können.

5.3.5 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 benutzerseitigen 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 essenziell für das Stateless-Cursor-Konzept.

5.3.6 Beispielhafte Anfrageverarbeitung

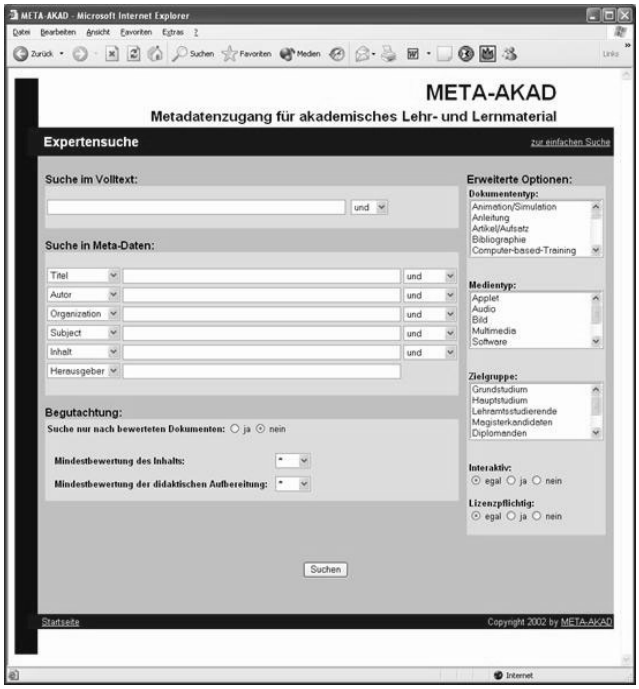


Abb. 5.11: Expertensuchschnittstelle

Auf die Anfrageverarbeitung soll nun anhand eines Beispiels näher eingegangen werden. Zunächst muss eine Query-Tree-Objekt-Struktur (siehe Abb. 5.10) 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 Processor Controller übergeben, der als zentraler Einstiegspunkt der Query-Engine-Komponente fungiert (siehe Abb. 5.11). Der Query Processor verarbeitet die Anfrage und erstellt, wie bereits erwähnt, eine temporäre Tabelle und lässt, wiederum durch die Query-Processor-Controller-Komponente, ein Stellvertreterobjekt („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 RSPCtrl übergeben werden, über den es möglich ist, eine Übersicht der sich qualifizierenden Tupel zu beziehen und über die gesamte Ergebnismenge zu iterieren (Abb. 5.12).

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

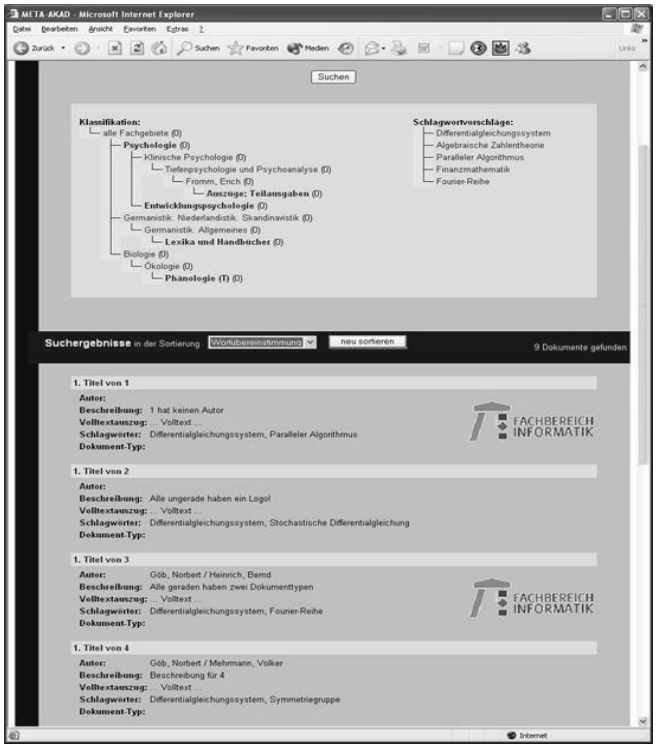


Abb. 5.12: Präsentation der Ergebnisliste

5.4 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 Anwen-

dungslogik der Dokumentenverwaltung. Sie hat zur Aufgabe, möglichst beliebige Suchanfragen entgegenzunehmen und ein entsprechendes Ergebnis zurückzuliefern. Es soll einfach möglich sein, aus den Metadatenattributen 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 datenbankseitig 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.

5.4.1 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. 5.13) 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. 5.13 nochmals graphisch dargestellt.

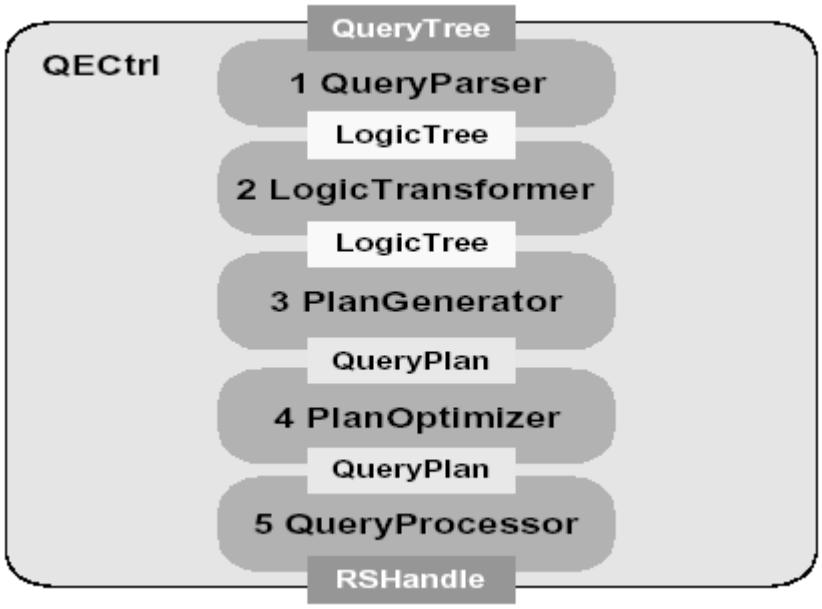


Abb. 5.13: Phasen der Anfrageverarbeitung

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.

5.4.2 Interne Repräsentation und Verarbeitung einer Anfrage

Die AV erhält die Anfrage in dem schon zuvor vorgestellten speziellen Format. Dieser Suchbaum („Query Tree“) bietet dem Anwender eine relativ einfache, schemaunabhängige 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 Query Tree 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 („Logic Tree“) einer Anfrage ist, wie die An-

frage 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 Logic Tree darstellbar. Diese Eigenschaft ist vor allem wichtig für spätere Transformationen der internen Darstellung.

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

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 Datenbanktabellen bzw. DB-Attributen sowie die Vorbereitung, Ausführung und Aufbereitung der Volltextsuche statt.

Die Metadatenattribute der erfassten Lehr-/Lern-Materialien (oder auch Ressourcen) werden in einem relationalen Schema verwaltet, wobei die einzelnen Merkmale in verschiedenen Tabellen gespeichert sein 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. 5.14 ersichtlich.

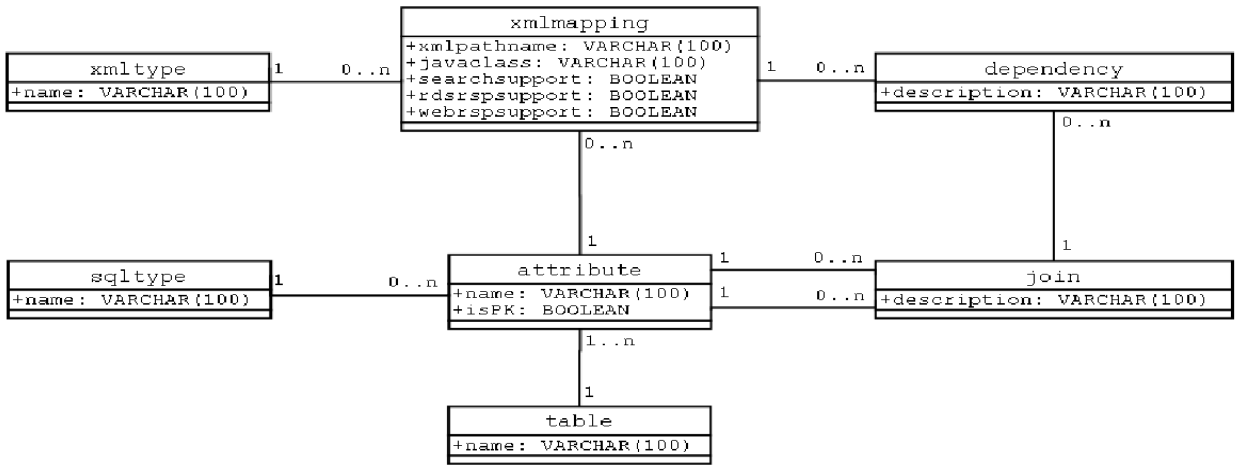


Abb. 5.14: Metamodell-Schema

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 predicate“) 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 Verbundbedingung 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 genau 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 gleiche Tabellen betreffen.
- Fasse alle Konjunktionen (AND) zusammen, welche ungleiche 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 und gemäß des Stateless-Cursor-Konzeptes darauf zugreifen zu können, was eine fortlaufende Nummerierung der Ergebnisse voraussetzt.

5.5 Ähnlichkeitsbasierte Suche

Die oben genannte Anfrageverarbeitung kann ferner durch eine ähnlichkeitsbasierte Suche weiter unterstützt werden. Diese Art von Suche findet man sehr häufig beispielsweise in den Bereichen E-Commerce, fallbasiertes Schließen, Wissensmanagement oder auch Text-Retrieval. Die Integration der ähnlichkeitsbasierten Suchen in ein Datenbanksystem (DBS) kann immense Vorteile bringen, da kostenintensive Funktionsaufrufe während der DB-internen Anfrageverarbeitung vermieden werden. 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. Ähnlichkeitsbasierte 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, HMR03].

5.6 Interoperabilität und OAI

Die Unterstützung von OAI-Funktionalität ist besonders im Verbund unterschiedlicher Digitaler Bibliotheken wichtig. Die Open Archives Initiative (OAI) entwickelte zu diesem Zweck Standards zur Verbesserung der Interoperabilität und der Möglichkeiten zum Austausch von Informationen unterschiedlicher Systeme im Bereich Digitaler Bibliotheken bzw. im Bereich von Katalog- oder Nachweissystemen. Kern dieser Standards besteht in der Spezifikation einer einfachen Anfragesprache und einer XML-basierten Repräsentation der Ergebnismengen. Die Flexibilität des Konzepts zur Ergebnisverarbeitung (Result Set Processor) machte eine Erweiterung zur Unterstützung von OAI in Meta-Akad sehr einfach. In Form eines einfachen Web-Service können OAI-konforme Anfragen gestellt werden und XML-Dokumente ausgetauscht werden. Die Realisierung geschah im Rahmen eines projektspezifischen Praktikums.

5.7 Redaktionsschnittstelle

Die Redaktionsschnittstelle soll die Möglichkeit bieten, komfortabel Einträge hinzuzufügen, zu ändern oder zu löschen. Dabei soll der Anwender durch eine umfassende Suchfunktionalität optimal unterstützt werden. Er soll über alle Metadatenfelder suchen und auch Anfragen über den Zustand der Erschließung stellen können, um den o. g. Prozess besser zu unterstützen. Im Rahmen eines projektspezifischen Praktikums wurde diese Komponente unter Berücksichtigung eines geeigneten Autorisierungskonzeptes entwickelt.

5.8 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.

5.8.1 Informationsextraktion

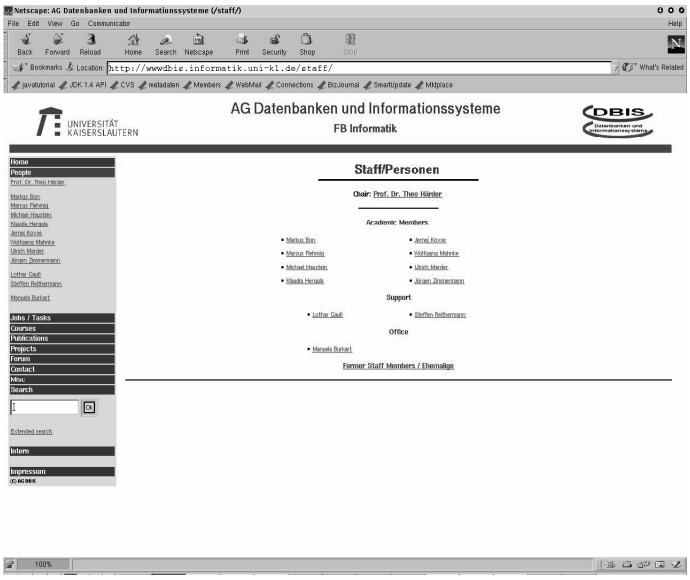


Abb. 5.15: Beispielseite

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. 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 vor-

kommen; zum anderen, dass die Regeln und Heuristiken, die verwendet werden, 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.

5.8.2 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.

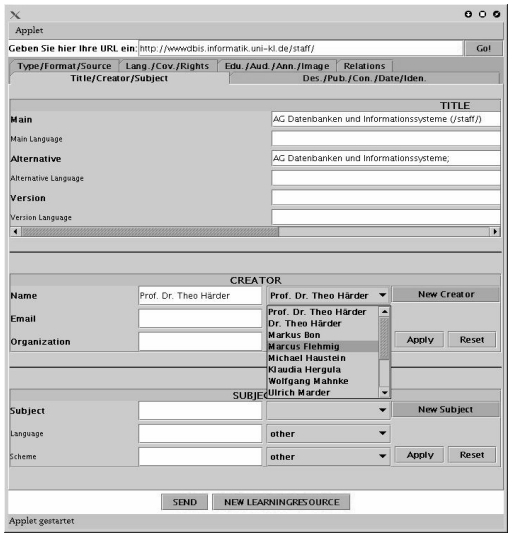


Abb. 5.16: Beispielanzeige des semi-automatischen Erschließungswerkzeuges

5.8.3 Beispiel

Das Beispiel in Abb. 5.15 zeigt ein HTML-Dokument mit Personennamen, die teilweise mit akademischen Titeln eingeführt werden. In diesem Fall werden mit Hilfe von Vor- und Nachnamenwörterbücher verschiedene Heuristiken angewendet und das Ergebnis als Vorschlag in den Feldern des semi-automatischen Erschließungswerkzeuges angezeigt (Abb. 5.16).

5.9 Automatische Erschließung

Nachdem erste Versuche bereits gezeigt hatten, dass ein statistischer Ansatz mit zuhilfenahme von Wortlisten zur automatischen Klassifikation nach der Regensburger Verbund Klassifikation (RVK)[29] 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 klassifi-

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

zierten 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].

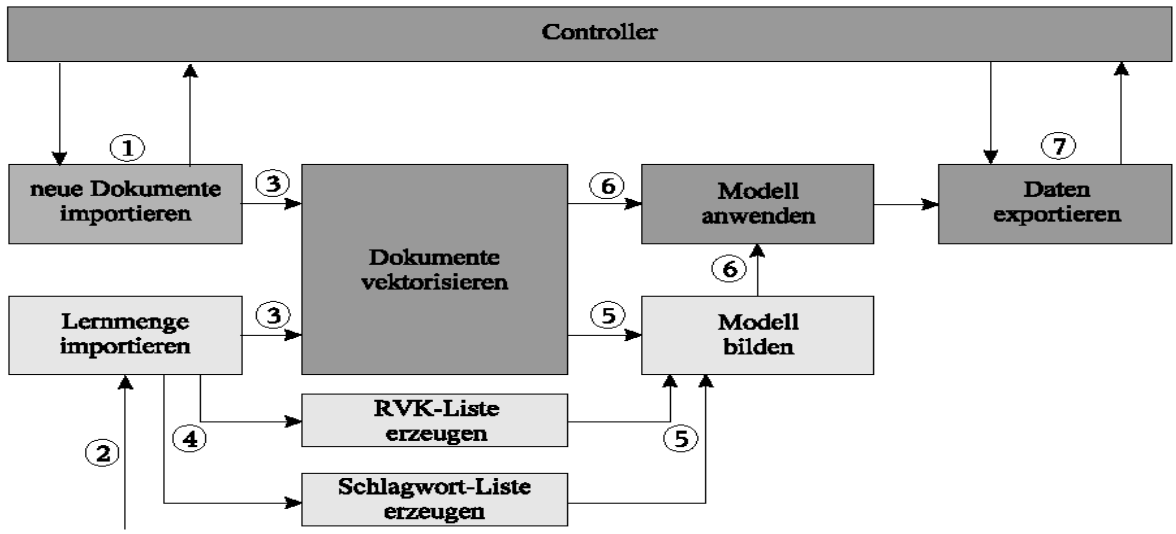


Abb. 5.17: Arbeitsablauf

5.9.1 Beschreibung des Klassifizierungs- und Beschlagwortungssystems

Abb. 5.17 zeigt den Arbeitsablauf des neuen Klassifizierungs- und Beschlagwortungssystems:

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.

3. **Dokumente vektorisieren**

   Um mit den bereits importierten Dokumenten arbeiten zu können, müssen diese erstmal in die Vektordarstellung überführt werden. In Abb. 5.18 ist die Konvertierung eines Dokumentes in eine Vektorrepräsentation dargestellt. Danach müssen die Wörter nach der Häufigkeit des Vorkommens in den einzelnen Dokumenten gewichtet werden.

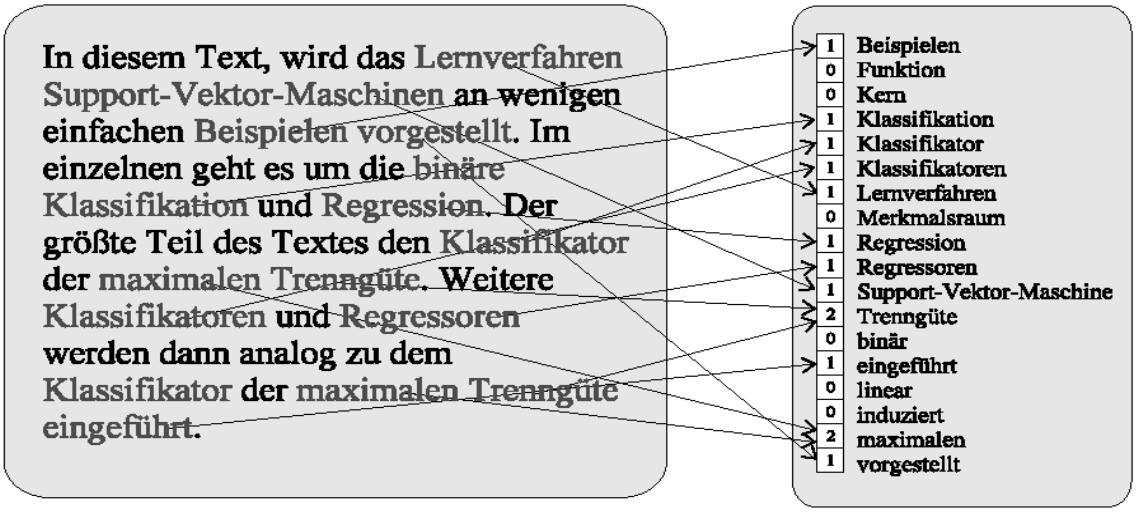


Abb. 5.18: Attribute-Wert-Darstellung eines Dokumentes

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.

In Abb. 5.19 ist die Überführung in einen höherdimensionalen Merkmalsraum (3-dimensional) 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 Modells gespeichert.

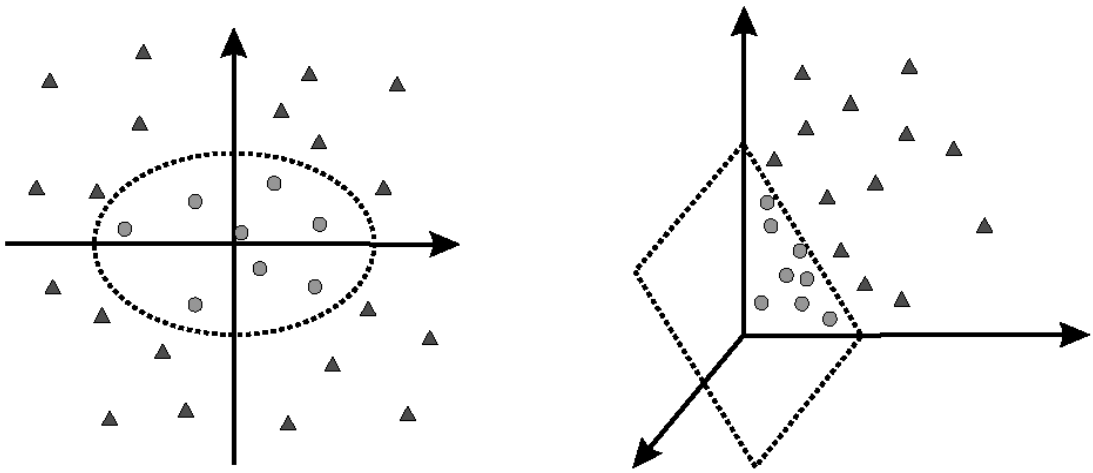


Abb. 5.19: Transformation in einen höherdimensionalen Merkmalsraum

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.

5.9.2 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:

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.

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.

Tab. 5.20: Vergleich von Ansatz 1 und 2

	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%

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].

6 Zusammenfassung

Die beschriebenen Arbeiten stellen aktuelle Forschungsthemen von großer praktischer Nützlichkeit dar. Im Rahmen des Projektes sind bereits eine Reihe von Werkzeugen und sicherheitsrelevanten Techniken entstanden, die im Fachbereichsalltag eingesetzt sowie laufend ergänzt und verbessert werden.

Ein großer Themenbereich konzentriert sich auf eine datenorientierte Sichtweise zur Verwaltung und Verarbeitung von Web-Dokumenten. Dazu haben wir mit dem Konzept des Unified View und der dazugehörigen Systemlösung einen umfassenden Ansatz zur Personalisierung im Rahmen einer Portal-Architektur entwickelt. Mit seiner Systemrealisierung, die sich von der Speicherungsschicht über die deklarative Bildung von Sichten auf die Dokumentenbasis bis zur Präsentation und dem Customizing der Dokumente für verschiedene Ausgabegeräte erstreckt, soll die praktische Tauglichkeit dieses Ansatzes nachgewiesen und evaluiert werden. Neben dem Gewinn aktueller Forschungsergebnisse bietet diese Vorgehensweise auch eine hervorragende Möglichkeit, Know-how im Bereich der Web-basierten Datenbankanwendungen und im Einsatz von XML zu erwerben.

Mit XCoP und dem dahinterstehenden Fragmentierungskonzept haben wir auch die dokumentenorientierte Sichtweise zur Handhabung und zum Aufbau von Web-Dokumenten untersucht und ferner Verbesserungen bei der Dokumentensuche, gestützt auf Techniken der Ähnlichkeitssuche hinsichtlich Struktur und Inhalt von Dokumenten, und ihre Integration in ORDBS untersucht.

7 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, TU Kaiserslautern, Feb. 2003.

Are01 Arends, M.: Möglichkeiten der Anwendungsintegration und Personalisierung Web-basierter Informationssysteme durch Portale am Beispiel eines Handelssystems für Warentermingeschäfte - Konzepte und Realisierung. Diplomarbeit, FB Informatik, TU Kaiserslautern, Sept. 2001.

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

Boh03 Bohler, T.: Spezifikation und Realisierung einer XML-Partition-Miner-Komponente für SHARX. Diplomarbeit, FB Informatik, TU Kaiserslautern, März 2003.

Büh02 Bühmann, A.: Möglichkeiten der Extraktion von Schemainformationen aus XML-Dokumenten und deren formale Beschreibung zur Verwendung in SHARX. Projektarbeit, FB Informatik, TU Kaiserslautern, Juli 2002.

Büh03 Bühmann, A.: Konzepte für die Einbettung bzw. Integration von XML und XQuery in eine Wirtssprache (Java). Diplomarbeit, FB Informatik, TU Kaiserslautern, April 2003.

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

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

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

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

FL01 Flehmig, M., Loeser, H.: Ansätze der Nutzung von Erweiterbarkeitsmechanismen zur Anwendungsintegration in ORDBS - Eine qualitative und quantitative Evaluierung. Tagungsband der GI-Fachtagung „Datenbanksysteme in Büro, Technik und Wissenschaft" (BTW'01), A. Heuer (Hrsg.), Informatik aktuell, Oldenburg, März 2001, Springer, 332-341.

Fle04 Flehmig, M.: A Scalable Component-Based Architecture for Online Services of Library Catalogs. Proc. 4th Int. Conf. of Web Engineering, Munich, Germany, July 2004, 396-401.

Fud03 Fudeus, S.: Generische Verwaltung von XML-basierten Daten in relationalen Datenbanksystemen im Rahmen des Projektes Meta-Akad. Projektarbeit, FB Informatik, TU Kaiserslautern, Dez. 2003.

Gau03 Gauß, B.: Skalierungskonzepte in datenintensiven, mehrschichtigen und verteilten Anwendungensarchitekturen (J2EE) und ihre quantitative Bewertung (Projekt Meta-Akad). Diplomarbeit, FB Informatik, TU Kaiserslautern, Mai 2003.

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

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

HMR03 Haustein, M., Mahnke, W., Ritter, N.: Demonstration of Index Techniques for Similarity-based Search in ORDBMSs. Technical Report of the 20th British National Conf. on Databases, Poster Papers, Coventry, UK, July 2003, 1-3.

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

HR01 Härder, T., Rahm, E.: Datenbanksysteme - Konzepte und Techniken der Implementierung. 2. Auflage, Springer, Aug. 2001.

Kal00 Kaldenbach, M.: Infrastruktur für den Austausch von versionierten und fragmentierten XML-Dokumenten im Web. Diplomarbeit, FB Informatik, TU Kaiserslautern, Okt. 2000.

Kir01 Kirmse, M.: Ablage und Verwaltung verschlüsselter Daten in einer relationalen Datenbank. Diplomarbeit, FB Informatik, TU Kaiserslautern, Aug. 2001.

KLS02 *Flehmig, M.*, Knüttel, H., Leiwesmeyer, B., *Ritter, N.*, Schmettow, M., *Weber, G.*: Virtuelle Lehre im Angebot der Universitätsbibliothek. Tagungsband der 16. DFN-Arbeitstagung über Kommunikationsnetze (LNCS - Proc. P-17), Düsseldorf, Mai 2002, 249-259.

Kre03 Krennrich, K.: Möglichkeiten der Identifikation und Notifikation von Änderungen bei Web-basierten Dokumenten durch Hash-Codes (Meta-Akad: Aktualisierungskomponente. Projektarbeit, FB Informatik, TU Kaiserslautern, Nov. 2003.

Loe99 Loeser, H.: iWebDB - Eine integrierte Web-Datenbank auf Basis objekt-relationaler DB-Technologie. Tagungsband der GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft" (BTW‘99), A. P. Buchmann (Hrsg.), Informatik aktuell, Freiburg, März 1999, Springer, S. 20-37.

Loe00a Loeser, H.: Shift it to the Server! - Let the Database Server Update Your Web Sites. Proc. 1st Int. Conf. on Web Information Systems Engineering (WISE 2000), Hongkong, June 2000, 46-50.

Loe00b Loeser, H.: Keeping Web Pages Up-To-Date With SQL:1999. Proc. Int. Database Engineering and Applications Symposium (IDEAS 2000), Yokohama, Japan, Sept. 2000, 219-223.

Loe00c Loeser, H.: ORDBS-Integration eines Index zur strukturbasierten Suche in XML-Dokumenten. Sept. 2000.

Loe01a Loeser, H.: Web-Datenbanken - Einsatz objekt-relationaler Datenbanksysteme für Web-Informationssysteme. Springer, 2001.

Loe01b Loeser, H.: ORDBS-gestützte Dokumentenaktualisierung für Web-Informationssysteme. Interner Bericht, April 2001.

LR99 Loeser, H., Ritter, N.: iWebDB - Integrated Web Content Management based on Object-Relational Database Technology. Proc. Int. Database Engineering and Applications Symposium (IDEAS’99), Montreal, Canada, Aug. 1999, 92-97.

LS00 Loeser, H., Surjanto, B.: Effizienter Informationsaustausch durch ORDBS-basiertes Web Content Management. Proc. Fachtagung CAD 2000, Berlin, März 2000, 51-67.

Lut04 Luttenberger, K.: Realisierung einer Mediator-Algebra zur verteilten Verarbeitung und Ausführung von XQuery-Anfragen im Projekt SHARX (Realisierung in J2EE). Projektarbeit, AG DBIS, TU Kaiserslautern, Aug. 2004.

Sch02 Schultz, A.: Spezifikation einer Registrierungskomponente für XML-Artefakte im Bereich personalisierter Web-Umgebungen. Projektarbeit, FB Informatik, TU 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, TU Kaiserslautern, Dez. 2001.

SRL00 Surjanto, B., Ritter, N., Loeser, H.: XML Content Management based on Object-Relational Database Technology. Proc. 1st Int. Conf. on Web Information Systems Engineering (WISE 2000), Hongkong, June 2000, 64-73.

Tuc03 Tuchbreiter, J.: Integration einer Volltextsuchmaschine in Meta-Akad. Projektarbeit, FB Informatik, TU Kaiserslautern, Okt. 2003.

Tuc04 Tuchbreiter, J.: Verarbeitungsmodell für die XML-basierte Integration heterogener Datenquellen in SHARX. Diplomarbeit, FB Informatik, TU Kaiserslautern, Juni 2004.

Wag02 Wagner, B.: XML-basierte Anfrageverarbeitung der Datenverwaltungskomponente im Rahmen des Projekts Meta-Akad. Projektarbeit, FB Informatik, TU Kaiserslautern, Jan. 2003.

Web02 Weber, C.: Realisierung einer automatischen Klassifizierungs- und Beschlagwortungskomponete im Rahmen des Projekts META-AKAD. Projektarbeit, FB Informatik, TU Kaiserslautern, April 2003.

Zen02 Zendel, O.: Leistungsbewertung von XML Publishing Frameworks. Projektarbeit, FB Informatik, TU Kaiserslautern, Feb. 2003.

Weitere Literatur

AAN01 Aboulnaga, A., Alameldeen, A. R., Naughton, J. F.: Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. Proc. 27th Int. Conf. on Very Large Data Bases, Roma, Sept. 2001, 591-600.

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

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

BD94 Bernstein, P. A., Dayal, U.: An Overview of Repository Technology. Proc. 20th Int. Conf. on Very Large Data Bases, Santiago, Chile, Sept. 1994, 705-713.

BGL+99 Baru, C. K., Gupta, A., Ludäscher, B., Marciano, R., Papakonstantinou, Y., Velikhov, P., Chu, V.: XML-Based Information Mediation with MIX. Proc. ACM SIGMOD Conf., Philadelphia, 1999, 597-599.

BKL+01 Borghoff, U., Koch, M., Lacher, M., Schlichter, J., Weißer:, K.: Informationsmanagement und Communities - Überblick und Darstellung zweier Projekte der IMC-Gruppe München, Informatik - Forschung und Entwicklung 16:2, Springer, 2001, 103-109.

Bla+03 Blanken, H., et al.: Intelligent Search on XML Data. LNCS 2818, Springer, 2003.

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

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

DAB+99 Deach, S., Adler, S., Berglund, A., Caruso, J., Milowski, A., Zilles, S.: Extensible Stylesheet Language (XSL) Specification, W3C Working Draft, April 1999.
http://www.w3.org/TR/WD-xsl.

Dav99 Davis J. R., : DataLinks - Managing External Data with DB2 Universal Database. IBM Corporation, Feb. 1999. http://www-4.ibm.com/software/data/pubs/papers/.

DFK+04 Diao, Y., Florescu, D., Kossmann, D., Carey, M., Franklin, M.: Implementing Memoization in a Streaming XQuery Processor. Proc. 2nd Int. XML Database Symposium, Toronto, Canada, August 2004, 35-50.

DFS99 Deutsch, A., Fernandez, M., Suciu, D.: Storing Semistructured Data with STORED. Proc. ACM SIGMOD Conf., Philadelphia, 1999, 431-442.

DM98 Dietinger, T., Maurer H.: GENTLE - General Network Training and Learning Environments. Proc. ED-Media98/ED-Telecom98, Freiburg, Deutschland, 1998, 59-74.

DOM02 DOM: Document Object Model. http://www.w3.org/DOM/, 2002.

FHP02 Frasincar, F., Houben, G.-J., Pau, C.: XAL - an XML algebra for query optimization. Database Technologies 2002, Proc 13th Australasian Database Conf. (ADC2002), Monash University, Melbourne, Victoria, Jan. 2002, 49-56.

FK99 Florescu, D., Kossmann, D.: A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. Technical Report, INRIA, France, 1999.

FLM98 Florescu, D., Levy, A., Mendelzon, A.: Database Techniques for the World-Wide Web: A Survey. ACM SIGMOD Record 27:3, 1998, 59-74.

FSC+03 Fernandez, M. F., Siméon, J., Choi, B., Marian, A., Sur, G.: Implementing Xquery 1.0: The Galax Experience. Proc. 29th Int. Conf. on Very Large Data Bases, Berlin, Germany, Sept. 2003, 1077-1080.

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

GV99 Grosso, P., Veillard, D.: XML Fragment Interchange. W3C Working Draft, June 1999. http://www.w3.org/TR/WD-xml-fragment.

GW97 Goldmann, R., Widom, J.: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. Proc. 23rd Int. Conf. on Very Large Data Bases, Athens, Greece, Aug. 1997, 436-445.

HKM01 Hasselbring, W., Koschel, A., Mester, A.: Basistechnologien für die Entwicklung von Internet-Portalen. Proc. GI-Fachtagung 'Datenbanksysteme in Büro, Technik und Wissenschaft' (BTW'01), A. Heuer, F. Leymann, D. Priebe (Hrsg.), Informatik aktuell, Oldenburg, 2001, Springer, 517-526.

HTML02 W3C's HyperText Markup Language Home Page. http://www.w3.org/MarkUp/, 2002.

IBM00 Informationsquellen für den Bereich Portale - URLs & andere Quellen. IBM eNews-Magazin, http://www-5.ibm.com/de/software/enews/elab/portale-linkliste.html, 2000.

Inf97 Informix Dynamic Server with Universal Data Option 9.1.X. Informix Software, Inc., http://www.informix.com/, 1997.

JAXP02 JAXP: Java API for XML Processing. http://java.sun.com/xml/jaxp/index.html, 2002.

JLST01 Jagadish, H. V. , Lakshmanan, L. V. S., Srivastava, D., Thompson, K.: TAX – A Tree Algebra for XML. Proc. 8th Int. Workshop of Database Programming Languages, Frascati, Italy, Sept. 2001, 149-164.

LCH+02 Lee, M.-L., Chua, B. C., Hsu, W., Tan, K.-L.: Efficient evaluation of multiple queries on streaming XML data. Proc. 2002 ACM CIKM Int. Conf. on Information and Knowledge Management, VA, USA, Nov. 2002, 118-125.

LPV00 Ludäscher, B., Papakonstantinou, Y., Velikhov, P.: Navigation-Driven Evaluation of Virtual Mediated Views. Proc. 6th Int. Conf. on Extending Database Technology (EDBT '99), Konstanz, Germany, 2000, 150-165.

Maz00 Mazzocchi, S.: Cocoon Publishing Framework 1 - User Guide. http://xml.apache.org/cocoon1/guide.html, 2000

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

MMM97 Mendelzon, A. O., Mihaila, G. A., Milo, T.: Querying the World Wide Web. Int. Journal on Digital Libraries, Volume 1, Number 1, 1997, 54-67.

MS03 Marian, A., Siméon, J.: Projecting XML Documents. Proc. 29th Int. Conf. on Very Large Data Bases, Berlin, Germany, Sept. 2003, 213-224.

MTL99 Merz, M., Tu, T., Lamersdorf, W.: E-Commerce - Technologische und organisatorische Grundlagen. Informatik Spektrum 22:5, Springer, 1999, 328-343.

NGT98 Naacke, H., Gardarin, G., Tomasic, A.: Leveraging Mediator Cost Models with Heterogeneous Data Sources. Proc. 14th Int. Conf. on Data Engineering, Orlando, Florida, USA, Feb. 1998, 351-360.

PC03a Peng, F., Chawathe, S. S.: XSQ – A Streaming XPath Engine. Technical Report CS-TR-4493 (UMIACS-TR-2003-62). Computer Science Department, University of Maryland, College Park, Maryland, USA, June 2003.

PC03b Peng, F., Chawathe, S. S.: XPath Queries on Streaming Data. Proc. 2003 ACM SIGMOD Int. Conf. on Management of Data, California, USA, June 2003, 431-442.

PV02 Papakonstantinou, Y., Vassalos, V.: Architecture and Implementation of an XQuery-based Information Integration Platform. IEEE Data Engineering Bulletin, Volume 25, Number 1, March 2002, 18-26.

Rah94 Rahm, E.: Mehrrechner-Datenbanksysteme: Grundlagen der verteilten und parallelen Datenbankverarbeitung. Addison-Wesley, 1994.

SA02 Sartiani, C., Albano, A.: Yet Another Query Algebra For XML Data. Proc. Int. Database Engineering & Applications Symposium, Edmonton, Canada, July 2002, 106-115.

RFC2396 Berners-Lee, T., Fielding, R., Irvine, U. C., Masinter, L: Uniform Resource Identifiers (URI): Generic Syntax. Request for Comments 2396, The Internet Engineering Task Force, 1998.

SAX02 SAX: Simple API for XML. http://www.saxproject.org/, 2002.

SBM98 Stonebraker, M., Brown, P., Moore, D.: Object-Relational DBMSs - Tracking the Next Great Wave. 2nd edition, Morgan Kaufmann Publishers, 1998.

Sch01 Schweizer, K.: Essay - Portal total. IBM eNews Magazin, http://www-5.ibm.com/de/software/enews/essay/2001-02-02-ess-1.html, 2001.

Smi00 Smith, R. M.: Talk to address the Internet Privacy and Profiling Senate Commerce Committee. Brookline, Massachusetts, USA, http://www.senate.gov/~commerce/hearings/0613smi.pdf, 2000.

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

STH+99 Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., De Witt, D., Naughton, J.: Relational Databases for Querying XML Documents: Limitations and Opportunities. Proc. 25th Int. Conf. on Very Large Data Bases, Edinburg, Scotland, 1999, 302-314.

Suc98 Suciu, D.: Semistructured Data and XML. Proc. 5th Int. Conf. of Foundations of Data Organization (FODO’98), Kobe, Japan, 1998.

TPC00 Transaction Processing Performance Council: TPC-W Benchmark Specification. http://www.tpc.org, 2000.

VGD+02 Viglas, S. D., Galanis, L., DeWitt, D. J., Maier, D., Naughton, J. F.: Putting XML query algebras into context. http://www.cs.wisc.edu/niagara/, 2002.

Wie92 Wiederhold, G.: Mediators in the Architecture of Future Information Systems. IEEE Computer 25:3, 1992, 38-49.

XDB02 XML:DB Initiative for XML Databases. http://www.xmldb.org/, 2002.

XIS01 XML Information Set. W3C Recommendation, http://www.w3.org/TR/xml-infoset/, 2001.

XLi00 XML Linking Language (XLink) Version 1.0. W3C Recommendation, http://www.w3.org/TR/xlink/, June 2001.

XML00 XML: Extensible Markup Language 1.0 (Second Edition). W3C Recommendation, http://www.w3.org/TR/2000/REC-xml-20001006, 2000.

XNS99 Namespaces in XML. W3C Recommendation, http://www.w3.org/TR/REC-xml-names/, 1999.

XSD00 XML Schema Part 1 - 3. W3C Recommendation, http://www.w3.org/xml/schema, 2001.

XSL01 XSL: Extensible Stylesheet Language. W3C Recommendation, http://www.w3.org/TR/2001/REC-xsl-20011015/, 2001.

XSLT99 XSLT: XSL Transformations. W3C Recommendation, http://www.w3.org/1999/REC-xslt-19991116, 1999.

XQL00 XML Query Requirements. W3C Working Draft, http://www.w3.org/TR/xmlquery-req, 2000.

ZR02 Zhang, X., Rundensteiner, E. A.: XAT: XML Algebra for the Rainbow System. Computer Science Technical Report Series, WPI-CS-TR-02-24, Worcester Polytechnic Institute, Worcester, Massachusetts, USA, 2002.