Erhard Rahm, Gottfried Vossen (Hrsg.)

Web und Datenbanken

Dpunkt-Verlag, Heidelberg, 2003

Vorwort der Herausgeber

Das World Wide Web, gelegentlich noch WWW abgekürzt, im täglichen Sprachgebrauch jedoch nur noch als »das Web« bezeichnet, hat die Welt nachhaltig verändert. Als Anfang der 90er Jahre eingeführter Internet-Dienst hat es dem Internet im Grunde erst zu seinem für jedermann erkennbaren Durchbruch verholfen. Wer heute irgendeine Form von Information benötigt, sucht danach oft zuerst im Web: Firmen stellen ihre Produktkataloge und Servicebroschüren dort ein, Dienstleister ihre Angebote; branchenspezifische Portale bündeln Informationen und Angebote bestimmter Bereiche (z.B. Reisen) und führen Preisvergleiche durch, Hochschulen und andere Ausbildungsstätten informieren über ihr Kursangebot, Vereine und Privatpersonen präsentieren sich, ihre Veranstaltungen bzw. ihre Hobbys. Darüber hinaus wird das Web in immer stärkerem Maße zur Geschäftsplattform: Man kann heute Neuwagen darüber konfigurieren und bestellen, Reisen aller Art buchen, Bankgeschäfte erledigen, Blumen bestellen, Fortbildungskurse absolvieren. Man darf inzwischen sogar davon ausgehen, dass eine Präsenz im Web für Firmen und Institutionen praktisch zwingend ist, dass eine solche für Einzelpersonen immer mehr zum Hobby wird, und dass man in wenigen Jahren über das Web auch auf Geräte in großem Stil wird zugreifen können, etwa die Motorsteuerung eines Automobils, die heimischen Jalousien oder die Überwachungskamera im Tresorraum einer Bank.

Alle diese Entwicklungen erfordern ein erhebliches Maß an Hardware- und Softwaretechnik. Hardwareseitig sind insbesondere leistungsstarke, ausfallsichere Server gefragt, welche eine hohe Anzahl von Zugriffen pro Zeiteinheit bedienen können; des Weiteren benötigt man eine umfassende Netzwerkinfrastruktur. In diesem Buch wird die Rede davon sein, was man *softwareseitig* benötigt. Die Vielfalt ist erheblich und reicht von einzelnen Algorithmen bis hin zu komplexen Systemen. Von zentraler Bedeutung sind daher seit Beginn der Entwicklung des Web *Datenbanken und Datenbanksysteme* gewesen. Man hat nämlich bereits früh erkannt, dass Inhalte, die mit einem Browser präsentiert werden, mit vergleichsweise einfachen Mitteln aktuell gehalten werden können, wenn sie im Moment des Zugriffs aus einer Datenbank erzeugt werden. Man hat ferner erkannt, dass die gigantische Menge an Inhalten für das Web nur dann einigermaßen effizient verwaltet werden kann, wenn man hier Datenbanktechniken einsetzt. Insofern ist die Verbindung von *Web und Datenbanken* – der Titel dieses Buches und gleichzeitig der Name eines im Jahre 2001 gegründeten Arbeitskreises

der Gesellschaft für Informatik (GI) – eine besonders wichtige und inzwischen auch eine besonders enge.

Allerdings sind die zahlreichen Fragestellungen selten einfach dadurch zu behandeln, dass man bekannte Techniken auf den neuen Kontext Web überträgt; dies betrifft Schema- und Sprachentwurf, effiziente Speicherung von Daten oder Webseiten, effizienten Zugriff darauf, angemessene Indexierung, sichere Verschlüsselung oder Leistungsmessung. Stattdessen sind Adaptionen nötig, die nicht selten zu völlig veränderten oder sogar zur Entwicklung von neuen Techniken geführt haben. Des Weiteren liegt einer Datenbank ja stets eine Modellbildung zugrunde, und auch auf dieser Ebene haben sich im Zusammenhang mit dem Web zahlreiche neue konzeptionelle Aspekte ergeben, von denen in diesem Buch wenigstens teilweise die Rede sein wird. Ein besonders wichtiger Modellierungsaspekt ist dabei, dass Web-Daten im Allgemeinen nicht mehr einer einheitlichen Strukturierung unterliegen, sondern dass selbst inhaltlich eng verwandte Daten unterschiedlich strukturiert sind, wenn sie aus verschiedenen Quellen stammen.

Abhilfe schaffen hier Auszeichnungssprachen, mit denen sich die inhaltlichen Komponenten von Daten in semantisch sinnvoller Weise markieren lassen. Weite Verbreitung hat in diesem Zusammenhang XML, die *eXtensible Markup Language*, erlangt, ein Sprachrahmen, dem in diesem Buch breiter Raum gewidmet wird. Datenbanksysteme müssen heute zumindest über Schnittstellen verfügen, über welche man XML-formatierte Daten importieren oder exportieren kann; daneben stellen manche Systeme inzwischen sogar ganz vom relationalen Datenmodell auf XML-Daten um. Auf konzeptioneller Ebene dient XML darüber hinaus zur Entwicklung von Vokabularen sowie Grammatiken, mit denen sich außer Daten auch Aktivitäten, Abläufe, Prozesse und Dienste beschreiben lassen, und dies nach Möglichkeit in einer Form, die wenigstens für eine bestimmte Branche Einheitlichkeit verspricht.

Der Einsatz von Datenbanken und XML hat auch wesentlichen Einfluss auf die Architektur von Web-basierten Informationssystemen. Hierbei geht es einmal darum, unterschiedlichste Anwendungen über einfach nutzbare Browser-Oberflächen sehr vielen Nutzer zugänglich zu machen und auf unterschiedlichen Servern vorliegende Datenbanken einzubinden. Des Weiteren fordern innovative Web-Einsatzmöglichkeiten immer stärker die Interoperabilität und Integration von Anwendungen und Datenquellen unterschiedlicher Herkunft. Diesen Anforderungen soll einerseits mit den rasant an Bedeutung gewinnenden *Web Services* auf Basis von XML-Technologien zum Datenaustausch und zur einheitlichen Registrierung von Diensten Rechnung getragen werden. Des Weiteren sind einheitliche Zugriffsschnittstellen auf die unterschiedlichsten Informationen in Form von Suchmaschinen, Portalen und Mediatoren anzubieten und weiterzuentwickeln. In diesem Bereich kann die Flexibilität von XML eine intelligente Kombination von Information-Retrieval- und Datenbanktechniken zur Verarbeitung von Texten, semistrukturierten Dokumenten und stark strukturierten Daten ermöglichen.

Aufbau des Buches

Zielsetzung des vorliegenden Buches ist es, einen fundierten Überblick über den aktuellen Stand und die zukunftsweisenden Entwicklungen aus der angesprochenen hohen Bandbreite von Themen im Bereich von Web und Datenbanken zu geben. Die 14 aufeinander abgestimmten Einzelkapitel, zu denen wir dem Leser an dieser Stelle eine Orientierungshilfe geben wollen, kommen von 32 ausgewiesenen Fachleuten. Die Beiträge gliedern sich in drei größere Teile:

I. Modellierung und Sprachen (Kapitel 1–3);
II. Architektur und Implementierung (Kapitel 4–10);
III. Anwendungen und existierende Systeme (Kapitel 11–14).

In Kapitel 1, *Semistrukturierte Datenmodelle und XML*, legen Benn und Langer die Grundlagen für die hier angestellten Betrachtungen zu Datenmodellen und zur Rolle von XML in diesem Zusammenhang. Sie erläutern, was man unter »semistrukturiert« versteht und wie man mit den syntaktischen Bestandteilen von XML die Modellierung von Daten selbstbeschreibend machen kann. Will man XML zwischen unterschiedlichen Quellen austauschen oder innerhalb einer bestimmten Community in einheitlicher Weise benutzen, muss man sich auf eine Grammatik für die betreffenden Dokumente einigen; hier kommen *Document Type Definitions* (DTDs) oder der neuere Vorschlag *XML Schema* zur Anwendung; letzterer ist Gegenstand des Kapitels 2 von Schöning und Waterfeld. Schemata für XML-Dokumente erlauben einen Abstraktionsgrad und eine Beschreibungsform, welche man im Kontext von Datenbanken in einem *Datenbankschema* vorfindet; bereits diese Analogie suggeriert, nach weiteren Anknüpfungspunkten für Datenbanktechniken und -konzepte im Umfeld von XML zu suchen. In Kapitel 3 beschreiben Lausen und May das *Anfragen, Ändern und Publizieren von XML*. Das Kapitel hat Anfragesprachen, Datenmanipulation und -transformation von XML-Daten zum Inhalt; dabei wird sowohl die Entwicklung der entsprechenden Konzepte beschrieben als auch eine Einführung in die gegenwärtig populärsten Sprachen wie XPath und XQuery in diesem Bereich gegeben.

Sodann beginnt Teil II über Architektur und Implementierung, wobei die Betrachtungen jetzt technischer werden. In Kapitel 4 beschreiben Kappel et al. zunächst die wesentlichen Alternativen und Konzepte zur *Architektur von Web-Informationssystemen*. Hier werden unter anderem der Aufbau gängiger Web-basierter Dienst- und Handelsplattformen sowie Techniken zur Anbindung von Datenbanken in Web-Anwendungen beschrieben. In Kapitel 5 stellen Klettke, Meyer, Retschitzegger und Unland die wesentlichen Alternativen zur *Speicherung von XML-Dokumenten* in Datenbanken vor. Dabei werden u.a. die unstrukturierte und strukturierte Speicherung von XML-Daten sowie die Verwendung herkömmlicher (objektrelationaler, relationaler, objektorientierter) Datenbanksysteme und der Einsatz eigens entwickelter XML-Speicherungsformen auf Basis eines Graphenmodells betrachtet. In Kapitel 6 widmen sich Sattler, Conrad und

Saake der Herausforderung von *Datenintegration und Mediatoren* und damit der Frage, mit welchen Methoden sich Daten aus unterschiedlichen Quellen zusammenführen und integrieren lassen, so dass sie für einen vorgegebenen Zweck in einheitlicher Weise zur Verfügung stehen. Weikum gibt in Kapitel 7 einen Überblick über Techniken des *Web Caching*. Hierbei geht es grundsätzlich um die Frage, wie sich Zugriffe auf Server und Seiten im Web durch Caching-Techniken, also lokales Zwischenspeichern, beschleunigen lassen. Helmer und Moerkotte behandeln in Kapitel 8 bekannte und weniger bekannte *Indexstrukturen für XML* und untersuchen, welche Datenbankindexe auch für grundsätzlich baumartig strukturierte XML-Dokumente anwendbar sind bzw. welche neuen Techniken hier benötigt werden. In Kapitel 9 geben Bruder, Heuer und Weber einen Überblick über den Stand der Technik bei *Suchmaschinen*; hier geht es um Methoden, nach denen Dokumente im Web zu einem vorgegebenen Suchbegriff gefunden werden, sowie darum, wie man in ein meist (zu) großes Suchergebnis Ordnung bringt. Schließlich wird in Kapitel 10 der Leser in das zunehmend Bedeutung erlangende Gebiet der *Web Services* eingeführt. Keidl, Kemper, Seltzsam und Stocker stellen konkrete Techniken vor, mit denen im Internet Dienste beschrieben, publiziert und angeboten, gefunden und schließlich zusammengesetzt und in Anspruch genommen werden können; dabei werden die heute in Gebrauch befindlichen Standards (wie SOAP oder UDDI) zur Illustration herangezogen.

In Teil III geht es um Anwendungen von Web und Datenbanken sowie verfügbare XML-Datenbanksysteme. In Kapitel 11 behandeln Rahm und Stöhr den Einsatz von Datenbanktechniken zur flexiblen *Auswertung des Nutzungsverhaltens von Websites*. Insbesondere wird für ein solches Web Usage Mining der Einsatz eines Web Data Warehouse diskutiert. *Web-basiertes Lernen* ist das Thema von Kapitel 12, in dem Jaeschke, Oberweis und Vossen den Stand der Dinge im Bereich des so genannten E-Learning bzw. des Lernens unter Rückgriff auf Webbasierte Plattformen darstellen. In Kapitel 13 geben Klettke und Meyer einen *Überblick über derzeitige XML-DBS*. Dabei werden einerseits die XML-Möglichkeiten etablierter Datenbanksysteme wie DB2, Oracle und SQL-Server betrachtet; andererseits erfolgt eine vergleichende Beschreibung XML-spezifischer Systeme wie Tamino und Excelon. Im letzten Kapitel 14 beschreiben Böhme und Rahm Techniken zum *Benchmarking von XML-Datenbanksystemen*, behandeln also das für Daten verwaltende Systeme wichtige Thema der Leistungsmessung.

Im Anhang folgen ein Glossar der wichtigsten Begriffe, ein Abkürzungsverzeichnis sowie Kurzbiographien der Autoren.

Das Buch richtet sich an Leser in Industrie und Hochschulen, die sich für technische Hintergründe des Web jenseits von hochglänzender Anwenderliteratur interessieren. Angesprochen sind Entwickler, Anwender, Studenten und Wissenschaftler, die sich in den behandelten Themengebieten einen aktuellen und fundierten Überblick verschaffen wollen. Es werden gewisse Vorkenntnisse vorausgesetzt, insbesondere über die Grundlagen des Web sowie in den Bereichen Datenbanken und Datenmodellierung.

Der bereits erwähnte GI-Arbeitskreis »Web und Datenbanken«, aus dessen Wirkungsfeld sich die an diesem Buch beteiligten Autoren im Wesentlichen rekrutieren, hat es sich zur Aufgabe gemacht, die zahlreichen Verbindungen zwischen dem Web und Datenbanken zu untersuchen und im Rahmen unterschiedlicher Aktivitäten aufzugreifen (genauer informieren kann man sich darüber – wie könnte es anders sein – im Web unter http://dbs.uni-leipzig.de/webdb). Das vorliegende Buch ist eine dieser Aktivitäten.

Wir sind an dieser Stelle zahlreichen Personen zu Dank verpflichtet, die zum Gelingen dieses Buchprojektes beigetragen haben. An erster Stelle gilt unser Dank den beteiligten Autoren, deren hohe Motivation und Einsatzbereitschaft es ermöglichten, einen für ein solches Projekt ungewöhnlich engen Zeitrahmen einzuhalten; das Buch ist de facto in einem Sechsmonatszeitraum entstanden, was andererseits für ein derart aktuelles Thema auch essenziell ist. Wir danken ferner Christa Preisendanz und René Schönfeldt vom dpunkt.verlag für die Unterstützung und verlagsseitige Betreuung bei der Erstellung des Buches.

Leipzig und San Ramon, Kalifornien, im August 2002
Erhard Rahm und Gottfried Vossen

Inhalt

Teil I: Modellierung und Sprachen

1 Semistrukturierte Datenmodelle und XML

Wolfgang Benn, Oliver Langer

Kurzfassung

Semistrukturierte Datenmodelle sind keine Ablösung bereits bestehender Modelle für Datenbanken, sondern integrierende Modelle für Daten, deren Strukturen unregelmäßig oder sogar unbekannt sind. Solche Daten sind im Internet häufig anzutreffen, und auch der Austausch von Daten zwischen heterogenen Anwendungen weist diese oder zumindest sehr ähnliche Probleme auf. Dieses Kapitel beschreibt die Eigenschaften semistrukturierter Daten und stellt exemplarisch zwei semistrukturierte Datenmodelle vor. Auf eines davon, nämlich auf XML, wird detaillierter bezüglich der Konzepte und Verarbeitungsmechanismen eingegangen, da hierin ein hervorragender Lösungsansatz zur Modellierung und zum Datenaustausch im Internet gesehen wird. Eine Diskussion über weitere Entwicklungen im Umfeld von XML schließt dieses Kapitel und leitet zu den nachfolgenden Kapiteln des Buches über.

1.1 Einleitung

Dieses erste Kapitel befasst sich mit einem relativ neuen Typus von Datenmodell: mit semistrukturierten Datenmodellen. Es soll eine Antwort auf die Frage geben, wo ein solcher Modelltyp im Kontext der bekannten und industriell eingesetzten Datenmodelle steht, welche theoretischen Grundlagen er besitzt und warum er gerade als *semi*strukturiert bezeichnet wird.

Um diesem Anspruch nachzukommen, beginnen wir bei der letzten Teilfrage – und wir tun dies am Beispiel des relationalen Modells. Dort finden wir eine hervorragende Strukturierung vor, die in jeder der unterschiedlichen Ebenen, die ein Datenbanksystem auf der Basis dieses Modells bedienen muss, ein Schema vorsieht. In diesem ist die Struktur der Daten so vollständig wie möglich beschrieben, und es dient dem Datenbanksystem ebenso als Interpretationsvorlage der Binärdaten wie zur Feststellung der Anwendungsanforderungen bei einer Anfrage.

Das heißt, wir haben es in einer relationalen Datenbank mit Strukturen zu tun, die (dem System und/oder auch dem Menschen, der sich ein Schema anschaut) vollständig bekannt sind. Und wir haben mit dem relationalen ein Modell, in dem die Strukturen der zu modellierenden Daten vollständig beschrieben sind.

Wenn auch der Begriff des Schemas für die Vorgängermodelle des relationalen Modells zum Teil gar nicht oder zumindest bei weitem noch nicht so konkret definiert war, so ist es dennoch eine Tatsache, dass auch diese Modelle (und die darauf aufbauenden Systeme) stets von der vollständigen Bekanntgabe der Struktur der zu verwaltenden Daten ausgingen.

Suchen wir nun nach Daten, die – aus Sicht der Informatik – eher unstrukturiert sind, so finden sich sehr schnell solche, zu deren Interpretation der Mensch derzeit noch viel besser in der Lage ist als die Maschine: Bilder, Texte und Klänge. Hier fällt es schwer, eine wirklich detaillierte Struktur anzugeben, die etwas über den Inhalt dieser Daten aussagt.

Im Internet sind im Regelfall beide Arten von Daten miteinander kombiniert, denn auf nahezu jeder Internetseite befinden sich Texte und Bilder – manchmal sogar Klänge und Video – und zunehmend werden die Seiten dynamischer, d.h., es befindet sich Information auf den Seiten, die in Maschinen zur Ausführung kommt und gar nicht zur Interpretation durch den Menschen gedacht ist.

Wir finden also eine Datenmischung, die mit einem herkömmlichen Schema nicht mehr sinnvoll zu beschreiben wäre, wollten wir derartige Daten in einer Datenbank verwalten. Gerade das ist aber notwendig, wenn sich der Aufwand zur Pflege und Verwaltung häufig wechselnder Internet-Seiteninhalte in vertretbaren Grenzen halten soll.

Noch wichtiger wird der Aspekt heterogener Daten und Datenstrukturen, wenn wir an die so genannten Web Services denken, die in zunehmendem Maße Dienste bzw. Dienstleistungen über das Internet anbieten. Hier geht es nicht nur darum, unterschiedliche Datenarten zu verwalten und zur Präsentation zusammenzustellen, sondern in diesem Bereich müssen Daten zwischen verschiedenen Anwendungen ausgetauscht werden können: Schließlich bietet nicht jede Anwendung einen Komplettservice für alle nur denkbaren Dienste, sondern es müssen oft mehrere Anwendungen sinnvoll zusammenarbeiten, um den vom Anwender via Internet aufgerufenen Service zu gewähren – und dabei ist es nicht immer notwendig (und zum Teil auch gar nicht möglich), allen beteiligten Anwendungen die Datenstrukturen aller beteiligten Systeme bekannt zu geben. Mit anderen Worten: Ein Datenmodell, welches die vollständige Kenntnis aller Strukturen voraussetzt, passt schlecht zu dieser Aufgabenstellung.

Wir schließen aus diesen Bemerkungen, dass ein semistrukturiertes Datenmodell in der Lage sein muss, Daten mit und ohne erkennbare Struktur gleichermaßen und gemischt modellieren zu können.

Mit Hilfe eines solchen Modells wird es natürlich auch möglich sein, Daten zu modellieren, die zwar inhaltlich gleich bzw. ähnlich, jedoch von ihrer Struktur her unterschiedlich sind. Wir können also die oft beklagte Heterogenität überwinden, die uns aus dem Kontext der Datenbankintegration (Stichwort: Schemaintegration), aus föderierten Systemen oder eben auch aus den heterogenen, lose gekoppelten Systemen des Internets bekannt ist.

Aus dieser Argumentation wird auch deutlich, dass ein semistrukturiertes Datenmodell nicht die Ablösung der bisher bekannten Datenmodelle anstrebt,

sondern im Gegenteil eher dazu eingesetzt werden soll, um die oft hervorragend optimierten, bestehenden Modelle zu integrieren bzw. einen Austausch von Daten, die diesen Modellen folgen, zu ermöglichen.

In den folgenden Abschnitten dieses Kapitels werden wir uns zunächst ganz allgemein mit den Eigenschaften semistrukturierter Daten befassen (Abschnitt 1.2), um dann beispielhaft zwei Datenmodelle – OEM und XML – vorzustellen, die zwar unterschiedliche Zwecke verfolgen, aber dennoch beide in den Bereich semistrukturierter Datenmodelle gehören (Abschnitt 1.3). Da XML für den Bereich Internet jetzt und in naher Zukunft von wachsender Bedeutung sein wird, geben wir in Abschnitt 1.4 eine kurze Einführung in die wesentlichen Konzepte dieses Modells bzw. dieser Sprache, um im folgenden Abschnitt (1.5) auf die Alternativen zur rechnerinternen Verarbeitung von XML-Dokumenten einzugehen. Abschließend diskutieren wir einige nach unserer Meinung noch fehlende Eigenschaften von XML, um wirklich für den Austausch von Internet-Daten bestens geeignet zu sein (Abschnitt 1.6). Eine kurze Zusammenfassung im Abschnitt 1.7 beschließt das Kapitel.

1.2 Eigenschaften semistrukturierter Daten

Datenbanksysteme sind in der Regel auf das Vorhandensein eines Schemas angewiesen, welches – analog zur Typdeklaration in einem Programm – zur Interpretation der Binärdaten auf dem Sekundärspeicher dient. Wenn wir es nun mit Daten zu tun haben, deren Struktur gar nicht oder nur zum Teil bekannt bzw. vorhanden ist, können wir mit großer Wahrscheinlichkeit davon ausgehen, dass diese Daten anderen Gesetzmäßigkeiten bezüglich der Stabilität ihrer Struktur unterliegen, als dies bei den klassischen Daten in Datenbanken der Fall ist. Wir fassen diese Eigenschaften unter dem Begriff der *Irregularität* zusammen und erläutern im Folgenden, welche Arten der Irregularität wir bei semistrukturierten Daten erkennen können.

1.2.1 Statische Irregularität

Als statische Irregularität bezeichnen wir in diesem Kapitel die Eigenschaft, dass Datenobjekte dauerhaft möglicherweise unterschiedliche Strukturen aufweisen, obwohl sie inhaltlich – also semantisch – gleiche oder zumindest sehr ähnliche Realweltobjekte repräsentieren. So können Attribute in einem Datenobjekt vorhanden sein und in einem anderen wiederum nicht bzw. es können gleich benannte Attribute jeweils unterschiedlichen Datentypen entsprechen.

Bekannt ist dieser Problemkreis aus dem Bereich der Sprache, wo wir Synonyme (Wörter, die gleiche Realweltobjekte mit unterschiedlichen Bezeichnungen versehen) und Homonyme (Wörter, die unterschiedliche Realweltobjekte mit gleichen Bezeichnungen versehen) erkennen und interpretieren können. Aber auch im Bereich der verteilten heterogenen Systeme (oft als föderierte Systeme bezeichnet),

stoßen wir bei der Schemaintegration seit vielen Jahren auf das Problem der statischen Irregularität.

Würde man für Daten mit starker statischer Irregularität klassische Datenbanksysteme verwenden, so entstünden viele Schemata mit jeweils nur schwacher Ausprägung und einem erheblichen Bedarf an Sekundärdaten zur Kennzeichnung von Datenobjekten als Repräsentanten gleicher oder ähnlicher Realweltobjekte.

1.2.2 Dynamische Irregularität

Neben einer statischen Irregularität können Strukturen in semistrukturierten Umgebungen auch im Verlauf der Zeit Änderungen unterliegen – wir bezeichnen dieses Phänomen hier als dynamische Irregularität. Dabei muss es sich nicht um kontinuierliche Änderungen handeln, sondern die Veränderungen können auch spontan geschehen.

Darüber hinaus können beide Formen der Irregularität vielgestaltig sein und neben dem Bereich der Objektstruktur auch die Veränderung von Typinformation zu bestimmten Attributen einschließen.

Dieses Problem ist aus den klassischen Datenmodellen für Datenbanken bereits bekannt und wird dort allgemein als Schema-Evolution bzw. in speziellerer Ausprägung als Versionierung bezeichnet.

1.2.3 Fehlende Schemainformation

Ein weiteres Merkmal der Irregularität ist, dass semistrukturierte Daten häufig gar keine Schemata verwenden. Im Unterschied zu den traditionellen Datenbankmodellen wird also überhaupt keine Strukturinformation an das verwaltende System vor der eigentlichen Verwendung der Daten gegeben. Stattdessen wird die Strukturinformation aus den Daten selbst entnommen. Man sagt auch, die Schemainformationen sind im Datenmodell selbst kodiert bzw. das Modell ist *selbstbeschreibend*. Ermöglicht wird diese Art der Selbstbeschreibung durch die Verwendung allgemeiner Graphen.

1.2.4 Pfadorientierter Datenzugriff

Wie aus der Graphtheorie bekannt ist, beschreibt ein Pfad den Weg von einem Startknoten zu einem Endknoten. Modellieren wir nun semistrukturierte Daten in dieser Form, so erkennen wir ein weiteres wesentliches Merkmal semistrukturierter Daten: Statt des aus relationalen Datenbanksystemen bekannten datensatzorientierten Zugriffs erfordern semistrukturierte Daten einen *pfadorientierten Zugriff*.

1.2.5 Zusammenfassung

Die Irregularität der Daten, das eventuelle Fehlen eines Schemas und die Verwendung allgemeiner Graphen sowie der daraus resultierende pfadorientierte Zugriff haben naturgemäß großen Einfluss auf die Performanz bei der Datenverwaltung. Beispielsweise gestaltet sich die Realisierung von Indexen schwieriger als gemeinhin üblich: Im Gegensatz zu Datensätzen, die durch Aggregation entstanden sind und einen wahlfreien Zugriff auf ihre internen Strukturen zulassen, sind Graphen Datenstrukturen, die keinen wahlfreien Zugriff auf Komponenten zulassen. Weiterhin stellt sich natürlich die Frage, worüber indexiert werden soll, wenn sich die Strukturen häufig ändern bzw. Regelmäßigkeiten gar nicht erst bekannt sind? Der Problematik des Indexierens widmet sich Kapitel 8 dieses Buches, so dass wir an dieser Stelle lediglich auf das Problem verweisen, nicht aber weiter darauf eingehen.

Abschließend fassen wir die wesentlichen Eigenschaften semistrukturierter Daten kurz zusammen:

- Die Struktur der Daten ist nur teilweise bekannt.
- Die Struktur der Daten ist irregulär und kann sich häufig ändern.
- Die Verwendung von Schemata ist nicht zwingend; wenn notwendig werden diese im Modell selbst kodiert, d.h., das Modell ist selbstbeschreibend.
- Das Datenmodell folgt dem Prinzip allgemeiner Graphen.
- Der Zugriff auf die Daten erfolgt pfadorientiert.

1.3 Semistrukturierte Datenmodelle

In diesem Abschnitt werden wir stellvertretend zwei Datenmodelle vorstellen, die im Kontext semistrukturierter Datenmodellierung von Bedeutung sind und deren Einsatzgebiet dennoch unterschiedlich ist. Eines – das so genannte *Object Exchange Model* (OEM) – ist ein rechnerinternes Modell, mit dem semistrukturierte Daten zur automatischen Verarbeitung in einem Rechner modelliert werden. Das andere Modell – die *eXtensible Markup Language* (XML) – ist zeichenorientiert und wird zur Modellierung semistrukturierter Daten verwendet, wenn die Lesbarkeit der Modellierung für Menschen gesichert und die Austauschbarkeit zwischen Rechnern mit unterschiedlicher interner Darstellungsweise von Daten gewährleistet werden soll. Beide Modelle sind graphbasiert und weisen die oben aufgezeigten Eigenschaften auf bzw. sind in der Lage, Irregularität zu modellieren.

1.3.1 OEM

Das *Object Exchange Model* kann quasi als eines der Urmodelle im semistrukturierten Bereich bezeichnet werden und hat sich als Referenzmodell bzw. als Basis für verschiedene Forschungsprojekte bewährt – z.B. Lore [MAGQ97] und TSIMMIS [CGHI94]. OEM wurde 1995 erstmals vorgestellt [PaGW95] und war

ursprünglich zur Integration heterogener Datenbestände vorgesehen. Es bildet einen Graph aus, in dem Objekte existieren, die einen Bezeichner, einen Typidentifikator, einen Wert und einen Objektidentifikator besitzen.

Hierbei nimmt der Bezeichner eine wichtige Rolle ein: Er dient zum einen – wie natürlich auch in anderen Modellen üblich – zur Identifizierung des Datenobjekts. Zum anderen besitzt er aber einen deskriptiven Charakter, in dem er die Bedeutung des Objekts angibt.

Mögliche Werte eines OEM-Objekts können atomare oder komplexe Datenobjekte sein. Bei atomaren Datenobjekten entspricht der Typidentifikator den aus Typsystemen bekannten einfachen und strukturierten Typen (integer, string, array etc.). Komplexe Datenobjekte hingegen können rekursiv unterstrukturiert sein, wobei der Typidentifikator auf eine Menge von Paaren aus (*attribute*, *object*)-Werten verweist. Hierbei entspricht *attribute* einem Bezeichner und *object* wiederum einem OEM-Objekt. Auf diese Weise repräsentiert der Bezeichner in OEM auf dem Niveau der eigentlichen Datenbeschreibung die *selbstbeschreibende* Eigenschaft dieses Modells.

Der Objektidentifikator wird in OEM in Analogie zu objektorientierten Datenbanksystemen benutzt und ermöglicht die eindeutige Identifizierung eines Datenobjekts. Im Unterschied zu seinem objektorientierten Pendant können OEM-Objektidentifikatoren aber den vordefinierten Wert null für *undefiniert* zugewiesen bekommen.

In der Terminologie allgemeiner Graphen lässt sich das OEM-Modell somit wie folgt beschreiben: Endknoten enthalten Werte der bereitgestellten Basistypen. Nicht-Endknoten verweisen über benannte Kanten auf Nachfolgeknoten.

Das OEM-Modell hat die Sichtweise auf semistrukturierte Daten bis heute mitgeprägt: Durch die graphbasierte Modellierung wird es den in Abschnitt 1.2.5 genannten Eigenschaften gerecht; darüber hinaus besitzt es im Vergleich zu relationalen, objektorientierten und objektrelationalen Datenmodellen eine dem Problemkreis adäquate Datenstruktur, die Einfachheit eines guten Modells und das benötigte hohe Maß an Flexibilität.

Allerdings offenbart sich im Vergleich zu den genannten Datenmodellen auch ein Nachteil: Bis heute existiert kein einheitliches und damit kein standardisiertes Datenmodell. Stattdessen sind etliche Derivate (beispielsweise Lore [MAGQ97], Strudel [FFLS00], Araneus [MAMM98]) mit jeweils individuellen Erweiterungen, speziellen Anfragesprachen und aufsetzenden Architekturen entstanden.

Man mag erwidern, dass bei einem rechnerinternen Datenmodell dieser Nachteil weitaus weniger schwerwiegend ist als bei einem Modell, welches speziell für den Austausch von Daten zwischen unterschiedlichen Umgebungen im Einsatz ist, aber gerade der Datenaustausch war ja eine der wesentlichen Zielsetzungen unserer Eingangsüberlegungen.

Damit kommen wir zur Vorstellung des zweiten, gerade für diesen Zweck eingesetzten Datenmodells, zu XML, welches durch bestehende Standards bzw. eine große Anzahl noch in Fluss befindlicher Standardisierungsbemühungen von dem Nachteil des Wildwuchses befreit bleiben soll.

1.3.2 XML

XML entstand aus den Erfahrungen mit der *Structured Generalized Markup Language* (*SGML*) und der ebenfalls zur Familie der Auszeichnungssprachen (*markup languages*) gehörenden *HyperText Markup Language* (HTML). Während sich mit SGML ein sehr gradliniger und wohlstrukturierter, jedoch komplexer und damit schwierig zu handhabender Standard entwickelt hat, weist HTML deutliche Schwächen auf, was die Trennung unterschiedlicher Dokumentinhalte angeht (z.B. Text, Verweise, Formatierungen, Skripte etc.). Außerdem ist HTML darstellungsorientiert und unterstützt damit die Trennung zwischen Dokumentinhalten und deren Präsentation ungenügend bis gar nicht.

Als Folge dieser Entwicklung gründete sich 1996 eine XML-Arbeitsgruppe unter der Schirmherrschaft des World-Wide-Web-Konsortiums (W3C) und es entstand die *eXtensible Markup Language* (XML) als stark vereinfachte, jedoch ebenso wohlstrukturierte Untermenge von SGML. Prinzipiell kann jedes XML-Dokument auch als entsprechendes SGML-Dokument angesehen werden.

Zehn Zielsetzungen wurden von der Arbeitsgruppe für das Design der Sprache aufgestellt, und insgesamt ist festzustellen, dass XML sowie die darum herum angesiedelten Werkzeuge diese Zielsetzungen erfüllen bzw. deren Erfüllung unterstützen. Die zehn Zielsetzungen sind:

1. XML soll einfach und unkompliziert im Internet verwendbar sein.
2. XML soll eine große Zahl von Applikationen unterstützen.
3. XML soll mit SGML kompatibel sein.
4. Programme, die XML-Dokumente verarbeiten, sollen einfach zu schreiben sein.
5. Die Anzahl optionaler Eigenschaften soll in XML so gering wie möglich gehalten werden (am besten gleich null).
6. XML-Dokumente sollen lesbar und (leidlich) verständlich sein.
7. XML sollte rasch definiert werden.
8. Das Design soll formal und prägnant sein.
9. XML-Dokumente sollen einfach zu erstellen sein.
10. Eine gewisse Knappheit in der Markierungsweise von XML steht nicht im Vordergrund.

XML stellt damit im Hinblick auf die Anwendbarkeit von Auszeichnungssprachen eine bedeutende Vereinfachung gegenüber SGML und bezüglich der Strukturierungsmöglichkeit von Dokumenten eine erheblich bessere Trennung verschiedenartiger Dokumentinhalte gegenüber HTML dar.

Auch XML ist ein graphbasiertes Datenmodell. XML-Dokumente bilden einen gerichteten zyklenfreien Graph – also einen Baum – aus und sind damit grundsätzlich hierarchisch strukturiert, wobei diese Strukturierung, ähnlich wie im OEM-Datenmodell, deutlich deskriptiven Charakter hat. XML-Elemente können jedoch trotz der Baumstruktur Elemente derselben oder einer anderen Hierarchie referenzieren, ohne mit ihnen direkt verbunden zu sein. Die Inhalte

der XML-Elemente selbst sind Texte oder liegen in einem Fremdformat vor (z.B. Bilder). Jedem Inhalt eines XML-Elements ist ein Bezeichner zugeordnet. Damit ist die Ähnlichkeit mit dem OEM-Datenmodell offensichtlich.

Anders als bei OEM liegt die Struktur eines XML-Graph in Textform vor (nicht nur die Inhalte) und besteht damit aus für Menschen verständlich bezeichneten Elementen. Durch diese Form ist XML kein rechnerinternes Modell zur Organisation von Daten, sondern dient vielmehr als Schnittstelle zwischen Mensch und System beziehungsweise als Austauschformat zwischen lose gekoppelten Systemen – wie sie, gemäß unseren Eingangsüberlegungen, im Internet generell anzutreffen sind.

Ein weiterer Unterschied zwischen XML und OEM ist die Ordnung innerhalb der Menge von Nachfolgern eines Elements, die im OEM fehlt. Wie wir in Abschnitt 1.4 sehen werden, ist diese Eigenschaft wichtig, um eine sinnvolle Strukturierung von Dokumenten zu realisieren.

Ebenfalls ist im OEM nicht vorgesehen, komplexe Datentypen durch Schemata explizit angeben zu können. Sie ergeben sich dort implizit aus dem selbstbeschreibenden Charakter des Graphmodells. Und bei strikter Berücksichtigung der Annahme, dass sich die Struktur der Daten häufig ändern kann sowie in der Regel unterschiedlich ausgeprägt ist (vgl. die Bemerkungen über Irregularität), erscheint die Verwendung von Schemata zunächst auch als hinderlich.

Wie allerdings verhält es sich mit dieser Annahme, wenn die Irregularität für eine Teilmenge der zu verwaltenden Daten letztendlich doch nicht zutrifft, d.h., wenn eine Teilmenge der Daten durchaus einem bestimmten Schema entspricht? Dann wird die optionale Definition und Verwendung von Schemata durchaus wünschenswert – und ist daher in XML auch gestattet. Daten können in XML also sowohl mit als auch ohne Vorgabe eines Schemas strukturiert werden. Es existieren sogar Überlegungen, Schemata aus vorhandenen Beschreibungen mittels einer Analyse zu erkennen, d.h. festzustellen, ob gewisse Daten im Bestand eine gemeinsame Struktur aufweisen (siehe z.B. [Suci01]).

1.3.3 Zusammenfassung

Zusammenfassend lässt sich sagen, dass die Stärken beider Modelle trotz großer Ähnlichkeiten dennoch unterschiedlich sind: OEM und ähnliche Modelle eignen sich insbesondere zur rechnerinternen Darstellung und späteren Verarbeitung semistrukturierter Daten – und setzen damit, wenn überhaupt eine Systemkopplung vorausgesetzt werden soll, eine erheblich engere Kopplung voraus, als dies bei XML der Fall ist. XML hingegen eignet sich als Austauschformat für lose gekoppelte, heterogene Systeme – insbesondere eben für das Internet. Abbildung 1-1 stellt diesen Zusammenhang dar.

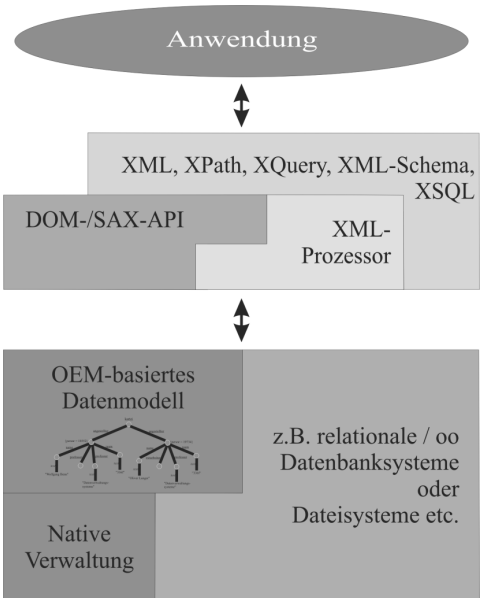


***Abb. 1-1:** : OEM- und XML-Datenmodell*

Die Abbildung zeigt, dass für die Verwaltung eines rechnerinternen semistrukturierten Datenmodells in vorhandenen Datenbanksystemen eine Übersetzung (*mapping*) dieses Modells auf das Datenmodell des Datenbanksystems – z.B. relationales, objektorientiertes, objektrelationales – notwendig ist. Wie in Kapitel 5 und 13 dieses Buches dargestellt wird, existieren jedoch auch Lösungen, die OEM-basierte Datenmodelle, ebenso wie XML, in einer speziell dafür konzipierten Datenbank nativ speichern. Motivationen hierfür stellen die in Abschnitt 1.2 genannten Eigenschaften dar, die sich zum Teil nicht gut mit den Eigenschaften relationaler, objektorientierter und objektrelationaler Datenbankmodelle vereinen lassen.

Unabhängig von der internen Repräsentation der Daten müssen jedoch an der Schnittstelle zwischen interner und externer Repräsentation semistrukturierter Daten in jedem Falle Funktionen zur Durchmusterung der Hierarchien bereitgestellt werden. Solche Funktionen umfassen entweder pfadorientierte Aufgaben, wie die Ermittlung der Kinder eines Elternelements, das Auslesen von Attributwerten eines Elements etc. oder das direkte Erkennen und Reagieren auf Markierungen im Dokument. Für XML wurden hierzu das *Document Object Model* (DOM) [HORS02] und das *Simple API for XML* (SAX) definiert, auf die wir im Abschnitt 1.5 näher eingehen.

1.4 Grundlegende Konzepte von XML

Im Sinne der Zielstellung dieses Buches, XML als besonders geeignetes Datenmodell für das Internet – und insbesondere auch für Datenbanken im Internet –

zu präsentieren, geben wir in diesem Abschnitt eine Kurzbeschreibung von aus unserer Sicht grundlegenden XML-Konstrukten. Diese ist mit Absicht unvollständig, da sich anderenfalls hier die komplette Spezifikation von XML in deutscher Übersetzung anschließen müsste. Der interessierte Leser sei für weitergehende Informationen auf das Internet – insbesondere die Seiten des W3C-Konsortiums (www.w3c.org) – stellvertretend für Bücher über XML auf [GoPr98] und für eine Kurzreferenz auf [Ecks99] verwiesen. Unser Abschnitt 1.4 gliedert sich in drei Hauptbereiche: in die Beschreibung logischer und physischer Strukturen des XML-Modells sowie in die Betrachtung von Konzepten, die insbesondere den semistrukturierten Charakter des Modells aufzeigen.

1.4.1 Logische Strukturen

Die hierarchisch oberste Struktureinheit in XML stellt das Dokument (*document*) dar, welches aus so genannten Elementen (*elements*) besteht. Dabei ist es wichtig zu wissen, dass ein Dokument mindestens aus einem Element zu bestehen hat – nämlich dem so genannten Wurzel- oder Dokument-Element. Der Inhalt (*content*) eines Elements besteht wiederum aus Elementen, wobei im Sinne einer Iterationsmöglichkeit Schachtelungen auftreten dürfen, nicht aber Rekursionen oder die Zugehörigkeit eines Elements als Inhalt unterschiedlicher Elternelemente. In Baumterminologie werden Elternelemente auch als Elternknoten (*parent*) und Inhaltselemente als Kindknoten (*child*) bezeichnet.

Bei einem Buch als XML-Dokument wären zum Beispiel *Titel*, *Kapitel*, *Überschriften*, *Abschnitte*, *Absätze* oder *Bilder* Elemente. Ein Buch enthält mehrere Kapitel, diese wiederum mehrere Abschnitte, und die ihrerseits wiederum Text und Grafiken. Die XML-Darstellung dieses Sachverhaltes zeigt Abbildung 1-2, in der die Elemente durch Markierungen (*tags*) bezeichnet werden. Jedes Element beginnt mit einem *start-tag* und schließt mit einem *end-tag*. Darin eingeschlossen finden sich die inhaltsgebenden Elemente für das durch *start-* und *end-tag* definierte Elternelement.

```
<buch>                                  Beginn Dokument-Element; Baumebene 0
  <titel>                                  Beginn Kind-Element; Baumebene 1
    Einführung in XML                             Inhalt Kind-Element: Text
  </titel>                                   Ende Kind-Element; Baumebene 1
  <kapitel>                                Beginn Kind-Element; Baumebene 1
    <ueberschrift>                         Beginn Kind-Element; Baumebene 2
      Historisches zu XML                         Inhalt Kind-Element: Text
    </ueberschrift>                          Ende Kind-Element; Baumebene 2
    <paragraph>                            Beginn Kind-Element; Baumebene 2
  XML wurde durch eine
      Arbeitsgruppe...                            Inhalt Kind-Element: Text
    </paragraph>                             Ende Kind-Element; Baumebene 2
  </kapitel>                                 Ende Kind-Element; Baumebene 1
  ...
</buch>                                  Ende Dokument-Element; Baumebene 0
```

Abb. 1-2: *Beispiel einer XML-Modellierung*

An diesem Beispiel können wir gleich mehrere Merkmale erkennen, die für das Design von XML festgelegt waren: Die Form ist für Menschen lesbar und verständlich. Die Modellierung liegt im Textformat vor und kann so mit allen gängigen Editoren dargestellt bzw. bearbeitet werden. Deutlich erkennbar ist auch die Unterscheidbarkeit von Elementen und deren eigentlichem Inhalt, da Elemente syntaktisch stets in spitzen Klammern eingeschlossen werden und sich damit klar vom Inhalt abheben. Zudem erhalten sie mnemonische, also sinntragende Bezeichner, so dass aus ihnen die Bedeutung der Strukturierung leicht abzulesen ist.

Der nachstehend aufgeführte Auszug aus dem XML-Standard [XML00], zeigt die Bildungsregel (*production*) für XML-Elemente, und der Leser wird auch ohne Detailkenntnisse der XML-Spezifikation erkennen, dass unser Beispiel aus Abbildung 1-2 dieser Bildungsregel entspricht[1].

```
[39]   element       ::=  EmptyElemTag | STag content Etag
[40]   STag          ::=  '<' Name (S Attribute)* S? '>'
[41]   Attribute     ::=  Name Eq AttValue
[42]   ETag          ::=  '</' Name S? '>'
[43]   content       ::=  CharData? ((element | Reference |
                          CDSect | PI | comment) CharData?)*
```

Abb. 1-3: *Syntaxvorgaben für XML-Elemente*

Bei genauem Lesen ist zu erkennen, dass die Bildungsregeln für Elemente neben der Verschachtelung auch die Angabe von Attributen zulässt. So kennzeichnet zum Beispiel das Attribut *nr*

```
<kapitel nr="1">
  Einführung in XML
</kapitel>
```

Abb. 1-4: *Erweiterung des Elements Kapitel um einen Parameter*

ein XML-Element vom Typ *kapitel*, dessen Attribut *nr* den Wert *1* enthält und dessen Inhalt dem Text »Einführung in XML« entspricht. Attribute bieten damit die Möglichkeit, ein Element mit zusätzlichen Eigenschaften zu versehen. Der Wert eines Attributs wird dabei immer in Anführungszeichen angegeben.

Diese Art der Zuordnung von Attributen zu Elementen ist vielen Lesern sicher bereits aus HTML bekannt, wo zum Beispiel in Tabellen Zeilenhöhen und Spaltenbreiten auf ähnliche Weise vorgegeben werden können. Die mangelnde Systematik der dort praktizierten Zuordnung ist aber an solchen Konstrukten, wie dem *style*-Attribut zu erkennen, welches – im Gegensatz zu den meisten übrigen Attri-

1. Die XML-Spezifikation ist in einer erweiterten Backus-Naur-Form angegeben. »S« steht für ein Leerzeichen – sog. white space; »?« bedeutet Option, d.h., das Zeichen kann auftreten, muss aber nicht; »*« steht für kein oder mehrmaliges Auftreten des voranstehenden Zeichens oder des voranstehenden Grammatikelements.

buten – selbst wieder eine Anzahl von Unterattributen besitzt und diese auch noch in abweichender Syntax angegeben werden müssen (Zuweisung der Werte durch »:« und nicht durch »=«).

In XML ist diese Zuordnung besser strukturiert, die Syntax ist exakt und einheitlich vorgegeben und die Angabe von Attributlisten zu Elementen genau spezifiziert. So sind die Attributtypen festgelegt (*StringType*, *TokenizedType* und *EnumeratedType*), es kann festgelegt werden, ob Attribute zwingend angegeben werden müssen oder nicht und ob ein Standardwert eingesetzt werden soll (*#REQUIRED*, *#IMPLIED* und *#FIXED*).

Wohlgeformtheit

Bislang haben wir die Bildung von XML-Dokumenten und -Elementen betrachtet. Der Begriff der *Wohlgeformtheit* eines XML-Dokuments besagt nun, dass ein solches Dokument und alle seine darin enthaltenen Elemente – sowie alle weiteren Bestandteile, die der XML-Spezifikation genügen sollen – auch tatsächlich dieser Spezifikation folgen. Mit anderen Worten: Ein so genannter XML-Prozessor prüft, ob die Bildungsregeln für Elemente, Attribute etc. in einem bestimmten ihm vorliegenden XML-Dokument so eingehalten wurden, wie es in der Spezifikation von XML festgeschrieben wurde. In unserem Beispiel aus Abbildung 1-2 kämen dafür die Bildungsregeln aus Abbildung 1-3 zur Anwendung und der Prozessor würde feststellen, dass es sich um ein wohlgeformtes Dokument handelt.

Mit Hilfe dieser Prüfung kann ein XML-Prozessor zwar feststellen, ob ein XML-Dokument der XML-Spezifikation entspricht, es ist aber nicht festzustellen, ob die in dem Dokument definierten Elemente dem entsprechen, was sie aussagen bzw. an Hand ihrer Struktur darstellen sollen. Dafür bedarf es einer weiteren Definition, in der bestimmt wird, welcher Grammatik die Elemente eines Dokuments zu folgen haben. (An dieser Stelle erkennen wir die Selbstbeschreibungs-Eigenschaft des semistrukturierten XML-Modells, nämlich Schemata – und das sind schließlich Bildungs- bzw. Interpretationsregeln – aus sich selbst heraus beschreiben zu können.)

Aus Gründen der leichteren Verständlichkeit haben wir bisher gesagt, dass ein XML-Dokument auf oberster Ebene der Hierarchie aus dem Wurzelelement besteht. Prinzipiell ist das auch korrekt, nur besteht ein wohlgeformtes XML-Dokument aus einer dreiteiligen Struktur, von der das Wurzelelement – als Stellvertreter des darunter angesiedelten Baumes – nur ein Teil ist. Davor steht der so genannte Prolog, und danach besteht die Möglichkeit, weitere Informationen in das Dokument zu bringen.

Damit ist der Prolog dem Wurzelelement vorangestellt und wird vom XML-Prozessor als erster Bestandteil der Dokumentregel ausgewertet. Er ist also durch seine Stellung prädestiniert, Angaben über die dem Dokument zugrunde liegende XML-Version und die Grammatik der Elemente zu enthalten: die *Dokumenttypdefinition* (*Document Type Definition* – DTD)

Fehlt die Angabe des Prologs – was möglich und zulässig ist –, beeinträchtigt das nicht die Wohlgeformtheit eines Dokuments, wohl aber dessen Gültigkeit

bezüglich einer Grammatik. Das Beispiel in Abbildung 1-2 stellt in diesem Sinne ein XML-Dokument dar, welches nur aus einem Wurzelelement besteht. Auf die Verwendung des Prologs werden wir später noch näher eingehen.

Gültigkeit

Die Angabe einer Dokumenttypdefinition hängt davon ab, ob das betreffende XML-Dokument einem bestimmten Dokumenttyp entsprechen soll oder nicht. Für das Beispiel in Abbildung 1-2 ist kein Dokumenttyp angegeben, und demzufolge werden die verwendeten Elementtypen mit ihrer Verwendung im Dokument eingeführt.

Ist jedoch eine Dokumenttypdefinition angegeben, müssen die im Dokument verwendeten Elementtypen aus dieser Grammatik herzuleiten sein. Erst wenn das für alle verwendeten Elemente geprüft und positiv beschieden wurde, entspricht das mit dem Wurzelelement beginnende XML-Element dem mit der DTD angegebenen Dokumenttyp. Es handelt sich dann nicht nur um ein wohlgeformtes, sondern auch um ein *gültiges* Dokument (gültig im Sinne der Bildungsregeln für die Elemente dieses Dokumenttyps).

Die Möglichkeit, Strukturvorgaben für die Gestaltung von Dokumenten anzugeben, existierte bereits in SGML. Sie wurde nach XML übernommen. Ähnlich wie Datentypen in Programmiersprachen oder wie Schemata in Datenbanksystemen bieten DTDs somit verbesserte Verarbeitungsmöglichkeiten der XML-Dokumente, was insbesondere deren Validierung, Vergleich und Durchmusterung betrifft.

Eine DTD besteht aus Bildungsregeln für Markierungselemente, die in einer so genannten *markup declaration* zusammengefasst sind. (In Anlehnung an andere generative Grammatiken stehen bei der Definition von DTDs auch Alternativen und die Angabe der Anzahl des Auftretens der Elemente zur Verfügung.) DTDs ermöglichen damit die Definition komplexer Elementtypen. Wir führen hier nachträglich die einfache Elementtypdeklaration für das Beispiel in Abbildung 1-2 ein:

```
<!ELEMENT buch (titel, kapitel*)>
<!ELEMENT titel (PCDATA #IMPLIED)>
<!ELEMENT kapitel (ueberschrift, paragraph*)>
<!ELEMENT ueberschrift (#PCDATA)>
<!ELEMENT paragraph (#PCDATA)>
<!ATTLIST kapitel nr CDATA #IMPLIED>
```

Abb. 1-5: *Elementtypdeklaration für Abb. 1-2*

Der Bezeichner eines Elements (in unserem Beispiel *buch* oder *kapitel*, *ueberschrift*, *paragraph*) wird auch *Typ* des Elements genannt. Ein Typ kann sich, wie aus dem Beispiel ersichtlich ist, durchaus aus mehreren Typen zusammensetzen. Der *Wert* des Elements ergibt sich dann aus dem zwischen Start- und Endemarke eingeschlossenen Inhalt – und zum anderen aus so genannten *Attributen*.

Die in Abbildung 1-5 gezeigte DTD besagt, dass XML-Dokumente vom Typ *buch* mit dem gleichnamigen Wurzelelement beginnen, einen Titel als Sohnelement besitzen müssen und Kapitel als weitere Kindelemente beinhalten. Diese wiederum enthalten eine Überschrift und Absätze, wobei beide Kindelemente des Kapitel-Elements als Inhalt so genannten einfachen, dem XML-Prozessor jedoch zugänglichen Text besitzen (PCDATA = *parsed character data*). Abschließend ist für das Kapitel noch eine Attributliste definiert worden, um dem Kapitel eine Kapitelnummer zuordnen zu können (wie in Abbildung 1-4 geschehen).

Interessant und wichtig anzumerken ist in diesem Zusammenhang, dass der Inhalt eines Elements auf diese Weise einem Muster bzw. einem Modell gleich vorgegeben werden kann. Hierzu dienen die sog. Inhaltsmodelle (*content model*).

In einem Inhaltsmodell wird deterministisch definiert, welche Kindknoten das aktuell definierte Element enthalten darf und in welcher Reihenfolge diese auftreten dürfen bzw. müssen. Ein Inhaltsmodell ist also eine verbindliche Teilbaumdefinition der Teilbaumebenen 0 und 1, die von einem endlichen Automaten fehlerfrei abgearbeitet werden kann. Sie enthält keine Angaben über Inhalte von Kindknoten. Die erste und zweite Zeile in Abbildung 1-5 stellen jeweils solch ein Inhaltsmodell dar.

Enthält ein Element lediglich Text oder Text und zusätzlich Kindknoten, so spricht man von gemischtem Inhalt (*mixed content*). Hierunter fallen also alle Knoten der Baumebene (Blatt-1), wenn diese Text enthalten – wie die Elemente *titel, ueberschrift* und *paragraph* in Abbildung 1-5.

Sinnvoll einzusetzen wäre ein solches Konzept im Kontext unseres Buchbeispiels, wenn wir für die Bildunterschriften festlegen wollten, dass diese stets mit der Abkürzung »Abb.« beginnen sollen. Wir definieren dann in der DTD den Elementtyp *abbildung* als

```
<!ELEMENT abbildung (#PCDATA | bild | unterschrift)>
```

und legen in diesem Fall erst im eigentlichen Dokument fest, wie die Elemente zueinander stehen und dass an der Stelle von #PCDATA die Abkürzung »Abb.« zu stehen hat. Eine zwingende Reihenfolge der Elemente, wie in einem Inhaltsmodell, kann nämlich bei gemischtem Inhalt nicht in einer DTD festgelegt werden.

Der entsprechende Dokumentteil sieht dann wie folgt aus:

```
<abbildung>
  <bild> ... </bild>
  Abb.                                                    Einfacher Text
  <unterschrift> ... </unterschrift>
</abbildung>
```

Abb. 1-6: *Mixed Content*

Der Prolog

Wie bereits erwähnt, ist der Prolog dem Wurzelelement vorangestellt und wird vom XML-Prozessor als erster Bestandteil der Dokumentregel ausgewertet. Damit ist die einfachste Methode, einem Dokument Bildungsregeln für Elemente voranzustellen, die Dokumenttypdefinition unmittelbar vor dem Wurzelelement in die gleiche Datei zu schreiben:

```
<?xml version="1.0"?>                                    Beginn des Prologs
<!DOCTYPE buch [                                            Beginn der DTD
  <!ELEMENT buch (titel, kapitel*)>
 <!ELEMENT titel (PCDATA #IMPLIED)>
  <!ELEMENT kapitel (ueberschrift, paragraph*)>
  <!ELEMENT ueberschrift (#PCDATA)>
  <!ELEMENT paragraph (#PCDATA)>
  <!ATTLIST kapitel nr CDATA #IMPLIED>
]>                                                  Ende der DTD und des Prologs
<buch>                                            Beginn des XML-Wurzelelements
  <titel>
    Einführung in XML
  </titel>
  <kapitel>
    <ueberschrift>
      Historisches zu XML
    </ueberschrift>
    <paragraph>
      XML wurde durch eine Arbeitsgruppe...
    </paragraph>
  </kapitel>
  ...
</buch>                                             Ende des XML-Wurzelelements
```

Abb. 1-7: *DTD und Dokument in einer Datei*

Sind DTDs jedoch lang oder möchte man eine bessere Strukturierung der Gesamtheit aller Dokumente erreichen (es sollen ja evtl. auch mehrere Dokumente einer DTD folgen), können DTDs extern definiert und dann im Prolog entsprechend referenziert werden. Bei der Referenzierung über

```
<!DOCTYPE buch SYSTEM "buch.dtd">
```

entspricht der Bezeichner *buch* dem Namen des Wurzelelementtyps. Über *SYSTEM* gefolgt von *buch.dtd* wird die Datei, in der sich die Elementtypdeklarationen befinden, in Form eines *Uniform Resource Identifiers* (URI) spezifiziert.

1.4.2 Physische Strukturen

Bislang haben wir uns mit einigen logischen Strukturen von XML und den Grundzügen der Vorgabe von Bildungsregeln für XML-Dokumente befasst. Das XML-Datenmodell beinhaltet aber auch Konzepte zur physischen Gruppierung und

Speicherung von XML-Elementen im Sinne der Modularisierung und Wiederverwendbarkeit. Diese Konzepte werden Einheiten (*entities*) genannt und sollen nachfolgend kurz erläutert werden.

Eingangs haben wir angemerkt, dass ein XML-Dokument als Baum dargestellt wird und damit auch klar ist, dass Kindknoten den Inhalt (*content*) ihres Elternknotens ausmachen, damit also auch nicht gleichzeitig Kinder zweier Elternknoten sein können. Mit anderen Worten: Eine Vernetzung zwischen Kindknoten, wie wir sie aus objektorientierten Modellen durch die *part-of*-Beziehung kennen, ist in XML nicht direkt nachzuvollziehen.

Andererseits ist es sinnvoll, mehrfach nutzbare Elemente zu separieren (modularisieren) und im Bedarfsfall zu referenzieren. Dies gilt für spezielle Zeichen (beispielsweise Sonderzeichen wie etwa Umlaute) ebenso, wie für komplexere XML-Elemente. Dabei ist es unerheblich, ob sich diese Einheiten innerhalb des aktuell betrachteten XML-Dokuments befinden (*internal entities*) oder nicht (*external entities*). Unterschiedlich ist lediglich die syntaktische Einbindung in das aktuelle XML-Dokument.

Wir werden in der folgenden Beschreibung den englischen Begriff *entity* im Text weitgehend vermeiden und mit den Bezeichnungen *Sonderzeichen* und *Modul* bzw. *Dokument* auf die spezielle Verwendung des *entity*-Konzeptes in XML abheben.

Sonderzeichen

Zum guten Verständnis dieses Konzeptes gehen wir hier zunächst auf die Anwendung des entity-Konzeptes zur Integration von Sonderzeichen in Text ein, da diese Arbeitsweise den meisten Lesern bereits durch HTML geläufig sein dürfte. Sonderzeichen in diesem Sinne sind alle Zeichen, die in der XML-Syntax verwendet werden und vom XML-Prozessor mit einer bestimmten Semantik assoziiert werden. Zur Verwendung im eigenen Text sind in XML die in Tabelle 1-1 angegebenen Sonderzeichen vordefiniert.

Entity-Referenz	(Sonder-)Zeichen
`&`	&
`<`	<
`>`	>
`&apos;`	’
`&quot;`	“

Tab. 1-1: *Entities zur Verwendung von Sonderzeichen*

Mit Hilfe dieses Konzeptes ist es auch recht einfach möglich, spezielle Sprachanpassungen durchzuführen, indem die besonderen Zeichen einer Landessprache als vordefinierte Sonderzeichen angegeben werden. Für die deutsche Sprache finden sich hier die Umlaute sowie die ß-Ligatur wieder:

Entity-Referenz	Umlaut
&auml;	ä
&Auml;	Ä
&ouml;	ö
&Ouml;	Ö
&uuml;	ü
&Uuml;	Ü
&szlig;	ß

Tab. 1-2: *Entities zur Verwendung von Umlauten*

Moduln, Dokumente und Dateien

Die tatsächliche Mächtigkeit des Konzeptes zeigt sich aber erst, wenn wir es auf XML-Dokumente, XML-Elemente und dem XML-Prozessor unbekannte, externe Dokumente in Kombination mit einem Referenzkonzept anwenden – was erlaubt und bezweckt ist. Das heißt, wir finden in diesem Konzept eine Analogie zu Referenzen in Programmiersprachen und Fremdschlüsseln im relationalen Datenmodell wie zu den Konzepten der Modularisierung und Wiederverwendbarkeit aus dem Software Engineering.

Nach dieser Bemerkung dürfte klar geworden sein, dass das aktuelle XML-Dokument selbst ein internes *entity* darstellt (das wir im Folgenden mit dem Begriff *Modul* bezeichnen) und eine extern definierte DTD ein externes *entity* ist (welches wir zur besseren Unterscheidung als *Dokument* bezeichnen).

Deklariert wird ein Modul im XML-Dokument durch:

```
<!ENTITY dtd "Document Type Definition">
```

Soll ein Modul innerhalb eines Elements referenziert werden, so beginnt die Referenz syntaktisch mit einem &-Zeichen, gefolgt von dem Modulnamen, und schließt mit einem Semikolon (Anmerkung: Rekursion ist auch auf diese Weise nicht erlaubt):

```
<bsptext>Die Kurzform der &dtd; lautet: DTD.</bsptext>
```

Bei Auftreten der Referenz *&dtd;* wird diese durch den in der Moduldeklaration (*entity declaration*) angegebenen Inhalt – hier den Text »Document Type Definition« – substituiert.

Handelt es sich bei einem Modul um ein für den XML-Prozessor *verständliches* Konstrukt, so erfolgt die Substitution direkt. Andernfalls müssen weitere Programme eingebunden werden. Letzteres trifft zum Beispiel zu, wenn eine Referenz zu einem externen Dokument führt, das eine Binärdatei ist, z.B. eine Bild- oder Klangdatei.

Aus diesem Grunde werden Dokumente in *übersetzbare* (*parsed entities*) und *nicht-übersetzbare* (*unparsed entities*) unterteilt. Nicht übersetzbare Dokumente können naturgemäß bezüglich des XML-Dokuments nur extern, d.h. in einer

anderen Quelle definiert sein. Folgerichtig entspricht die Deklaration einer Referenz auf diese Quelle:

```
<!ENTITY dombaum SYSTEM "D:/grafiken/dombaum.tif" NDATA tiff>
```

SYSTEM bezeichnet wie schon im Beispiel der externen DTD, dass es sich bei *dombaum* um ein externes Dokument handelt, welches unter der sich anschließenden Adresse aufzufinden ist. Durch das reservierte Wort *NDATA* und *tiff* wird dem XML-Prozessor mitgeteilt, dass es sich um ein nicht übersetzbares Dokument vom Typ *tiff* handelt.

Wird NDATA nicht angegeben – wie im folgenden Beispiel – so handelt es sich um ein extern definiertes, aber übersetzbares Dokument:

```
<!ENTITY kapitel1 SYSTEM "D:/xml-buch/kapitel01.xml">
```

Bei Auftreten der Referenz *&kapitel1;* wird der über »D:/xml-buch/kapitel01.xml“ spezifizierte Text vom XML-Prozessor eingefügt und anschließend am Beginn der Substitution mit der Übersetzung fortgefahren.

Auf diese Weise lassen sich umfangreiche XML-Dokumente auf mehrere Quellen verteilen und man hat umgekehrt die Möglichkeit, Dokumente je nach Bedarf aus Einzelteilen zusammenzusetzen. Im nachfolgenden Beispiel wird dies veranschaulicht, indem die Kapitel des Buches zu externen Dokumenten gemacht wurden und das XML-Dokument *buch* lediglich eine semantische Klammerung seiner Bestandteile darstellt:

```
<?xml version="1.0"?>
<!DOCTYPE buch SYSTEM "buch.dtd">
<buch>
  <titel>Web und Datenbanken</titel>
    &einleitung;
    &kapitel1;
    &kapitel2;
          ...
</buch>
```

Abb. 1-8: *XML-Dokument als Klammer externer XML-Dokumente*

In der DTD *buch.dtd* sind dann natürlich u.a. auch die folgenden Dokumente definiert:

```
<!ENTITY einleitung SYSTEM "D:/xml-buch/einleitung.xml">
<!ENTITY kapitel1 SYSTEM "D:/xml-buch/kapitel01.xml">
<!ENTITY kapitel2 SYSTEM "D:/xml-buch/kapitel02.xml">
```

Abb. 1-9: *Erweiterte DTD für das Beispiel in Abb. 1-8*

1.4.3 Weitere Konzepte

Neben den bislang besprochenen Konzepten zu Aufbau und Verifikation von XML-Dokumenten gibt es noch weiterführende Konzepte, die XML-Dokumente besonders flexibel machen bzw. deren Bearbeitung und/oder Interpretation durch Anwendungen bzw. menschliche Leser unterstützen.

Diese Konzepte sind zum einen Verarbeitungsanweisungen (*processing instructions*), zum anderen der so genannte CDATA-Bereich und schließlich Kommentare. Alle drei sind direkter Ausdruck der semistrukturierten Eigenschaften des Modells, denn es handelt sich in der Terminologie der Graphen um Endknoten des XML-Baumes, die durch den XML-Prozessor uninterpretierte Information enthalten und deren Inhalt zur Verarbeitung an andere Systeme oder den lesenden Menschen weitergereicht wird.

Verarbeitungsanweisungen sind dabei Anweisungen für Anwendungsprogramme oder auch für den jeweils verwendeten XML-Prozessor, die das betreffende Dokument verarbeiten und individuelle Funktionalitäten bei der Bearbeitung von XML-Dokumenten zum Einsatz kommen lassen. Verarbeitungsanweisungen werden direkt an die als Ziel (*PITarget*) angegebene Anwendung durchgereicht und nicht vom XML-Prozessor interpretiert (sofern nicht der Prozessor selbst das Ziel ist, denn auch dieser kann natürlich als Anwendung angesehen werden, wenn er Funktionalitäten enthält, die über den Standardumfang hinausgehen und die andere XML-Prozessoren deshalb evtl. nicht enthalten – ein Beispiel für genau diesen Spezialfall ist SAX, vgl. Abschnitt 1.5.2).

Syntaktisch beginnen Verarbeitungsanweisungen mit der Zeichensequenz **<?--** gefolgt von einem Bezeichner. Es schließt sich eine beliebige Zeichenkette an, die lediglich den Teilstring **?>** nicht enthalten darf, da dieser das Ende der Verarbeitungsanweisung markiert.

Im nachfolgenden Beispiel wird eine Verarbeitungsanweisung verwendet, die das Programm »MyApplication« veranlassen soll, an der entsprechenden Stelle im XML-Dokument das aktuelle Datum im angegebenen Format einzufügen.

```
<text>
  Dieses Dokument wurde am
    <?– MyApplication(insertDate format:"DD.MM:YYYY") ?>
  erstellt.
</text>
```

Abb. 1-10: *Beispiel einer Verarbeitungsanweisung*

Ein weiteres Konzept, die so genannte CDATA-Section, erlaubt es, Text, der XML-Markierungen enthält, in einem Dokument einzufügen, ohne dass diese Markierungen als solche vom XML-Prozessor interpretiert werden. Diese Art des Ignorierens von Endknoten im XML-Baum eignet sich insbesondere dazu, in XML-Dokumenten selbsterklärende Passagen einzufügen.

Syntaktisch beginnt die CDATA-Section mit der Markierung **<!**, gefolgt von der Schlüsselsequenz **[CDATA[**. Es folgt der vom XML-Prozessor zu ignorierende Text und schließlich die Endemarkierung (*CDEnd*) in Form von **]]>**.

Handelte es sich bei diesem Buch nicht um ein konventionell gedrucktes Werk, sondern um ein XML-Dokument, so müsste die Passage um das Beispiel der Abbildung 1-2 herum im Urtext dieses Dokuments wie folgt dargestellt werden (zur Vereinfachung hier ohne alle Angaben zur Formatierung, ohne Erklärungen und ohne Abbildungsunterschrift gezeigt).

```
<paragraph>
... Darin eingeschlossen finden sich die inhaltsgebenden Elemente für das durch start- und
end-tag definierte Elternelement.
</paragraph>
<![CDATA[
<buch>
  <titel>
    Einführung in XML
  </titel>
  <kapitel>
    <ueberschrift>
      Historisches zu XML
    </ueberschrift>
    <paragraph>
      XML wurde durch eine Arbeitsgruppe...
    </paragraph>
  </kapitel>
  ...
</buch>
]]>
<paragraph>
An diesem Beispiel können wir gleich mehrere Merkmale erkennen, die für das Design von
XML festgelegt waren:...
</paragraph>
```

Abb. 1-11: *Beispiel einer CDATA-Section*

Abschließend sollen die Kommentare erwähnt sein, die syntaktisch mit **<!--** begonnen und **-->** beendet werden. Wieder bezogen auf den Gedanken, dass auch Kommentare vom XML-Prozessor uninterpretierte Endknoten darstellen, ist in diesem Fall der das Dokument lesende Mensch das *Zielsystem*, an welches der Knoteninhalt weitergereicht wird.

1.4.4 Zusammenfassung

In diesem Abschnitt sind wir auf die uns wesentlich erscheinenden Eigenschaften und Konstrukte von XML eingegangen. Für weitere einführende Informationen

sei nochmals auf [GoPr98] verwiesen und als Referenz für Detailinformationen ist die aktuelle Ausgabe des Standards [XML00] in jedem Falle empfehlenswert.

Ohne auf jedes Detail der Spezifikation einzugehen, dürfte deutlich geworden sein, dass mit XML ein sehr systematisches Modell zur Strukturierung von Dokumenten zur Verfügung steht. Die Erstellung von Strukturen kann durch Angabe von Bildungsregeln in einer Dokumenttypdefinition vorgegeben und kontrolliert werden. Strukturen bzw. Teile davon können mit Attributen versehen werden, es können Teilstrukturen oder auch externe Daten referenziert werden, und es sind Eigenschaften enthalten, die es erlauben, externe Anwendungen mit Informationen zu versorgen, um die Verarbeitung von Dokumenten auch dann zu gewährleisten, wenn sie Passagen enthalten, die der XML-Prozessor nicht selbst verarbeiten kann. Damit ist XML ein idealer Standard für den Austausch von Web-Daten.

Im Abschnitt 1.3 wurde jedoch bereits gesagt, dass XML kein rechnerinternes Modell und der primäre Interpret von XML-Dokumenten der Mensch ist. Wollen wir XML-Dokumente dennoch maschinell verarbeiten, benötigen wir eine effiziente rechnerinterne Darstellung und eine Schnittstelle, die es uns erlaubt, in Anwendungsprogrammen auf Inhalte eben dieser internen Repräsentation zuzugreifen. Diesen Schnittstellen widmet sich der nächste Abschnitt unseres Kapitels.

1.5 Zugriff auf XML-Dokumente

In diesem Abschnitt geben wir eine Einführung in das *Document Object Model* (*DOM*) und das so genannte *Simple API for XML* (*SAX*). Bei DOM und SAX handelt es sich um Definitionen zum Zugriff auf interne Repräsentationen von XML-Dokumenten. DOM- bzw. SAX- Implementierungen sind also Bibliotheken (*Application Programming Interface – API*), die Funktionen bereitstellen, mit deren Hilfe Anwendungsprogramme auf rechnerintern repräsentierte XML-Dokumente zugreifen können. DOM und SAX stehen damit in der folgenden Verarbeitungskette:

Ein XML-Prozessor führt den Parserprozess eines XML-Dokuments durch, d.h., er führt die lexikalische Analyse der Markierungen und die Prüfung auf Wohlgeformtheit, evtl. auch auf Gültigkeit, durch und aktiviert möglicherweise auch noch die Ausführung weiterer Funktionen. Das Ergebnis des XML-Prozessors ist ein so genannter Parserbaum (*parse tree*), der bei Verwendung von DOM in eine rechnerinterne Datenstruktur überführt wird, die wiederum einen so genannten DOM-Baum repräsentiert. Bei der Verwendung von SAX hingegen wird der Parserbaum direkt während dessen Erzeugung einer semantischen Markierungsinterpretation unterzogen, womit SAX bereits an dieser Stelle als speziell ausgeprägter XML-Prozessor zu erkennen ist. Über die in DOM und SAX definierten API-Funktionen kann danach auf den Inhalt interner Datenstrukturen zugegriffen werden. Auf Grund der unterschiedlichen Philosophie beider Ansätze spricht man bei DOM von einem *baumorientierten* (*tree-based*) und bei SAX von einem *ereignisorientierten* (*event-based*) Interface.

Während DOM den Standardisierungsbemühungen des W3C unterliegt [DOM02], handelt es sich bei SAX um eine nicht standardisierte, informelle Spezifikation, die ursprünglich von David Megginson ausgearbeitet worden und aktuell im Internet unter http://www.saxproject.org zu finden ist. Wir werden nachfolgend beide Bibliotheken kurz beschreiben und jeweils auf die bereits erwähnten Unterschiede eingehen.

1.5.1 Das Document Object Model (DOM)

Das vom W3C-Konsortium spezifizierte Document Object Model ist ein plattform- und sprachunabhängiges Modell zur maschinellen Auswertung und Manipulation einer rechnerinternen Darstellung von XML-Dokumenten.

Während die bisherigen Bemühungen, Programmfunktionalität in Web-Dokumenten einzubinden, durchaus sprach- und zum Teil sogar herstellerspezifisch waren, wurde durch DOM eine Plattform geschaffen, die generisch ausgelegt ist, dessen Schnittstelle sich jedoch an bestehende Sprachstandards anbinden lässt – etwa an ECMAScript, einen Industriestandard aus JavaScript und JScript, oder auch an Java selbst. Auf diese Weise wird es möglich, die leidlich bekannten Differenzen bei der Interpretation von Skripten in Browsern unterschiedlicher Hersteller endlich zu überwinden. DOM liegt derzeit in der dritten Erweiterung vor.

Das Document Object Model verwendet eine objektorientierte Sicht der repräsentierten Dokumente. Demzufolge wurde für die Definition der DOM-Schnittstelle (DOM-API) die aus dem CORBA-Standard 2.2 der *Object Management Group* (OMG) bekannte Spezifikationssprache *Interface Definition Language* (IDL) übernommen. Auf diese Weise können bereits existierende Sprachanbindungen der CORBA-IDL an Programmiersprachen wiederverwendet werden.

Ein wesentlicher Grundzug des Document Object Model ist die Unabhängigkeit der rechnerinternen Darstellung von dem Modell des DOM-Baumes. Ein Nutzer der DOM-Schnittstelle kann sich – in Analogie des ihm möglicherweise bekannten XML-Baumes – auf eine Verarbeitung seines Dokuments entsprechend einem DOM-Baum verlassen. Es bleibt ihm sinnvollerweise verborgen, dass dieser Baum intern möglicherweise gar nicht als Baum realisiert wurde. Damit ist klar, dass DOM keine Binärkompatibilität zwischen Maschinen schafft, sehr wohl aber eine Sourcecode-Kompatibilität zwischen gleichen Sprachanbindungen bietet.

Betrachten wir nun kurz einige Details des Document Object Model. Ein XML-Dokument wird als DOM-Baum repräsentiert, der aus so genannten Knotenobjekten besteht. Jeder DOM-Baum besitzt – ganz entsprechend einem XML-Dokument – einen Wurzelknoten, von dem ausgehend alle Nicht-Wurzelknoten, welche die Subelemente vertreten, erreichbar sind. Jedes Knotenobjekt enthält Attribute, die über die Schnittstellen dieses Objekts abfragbar sind. Attribute sind z.B. der Name des Dokuments oder eines Elements, die Liste von Attributen eines Elements oder die Liste der enthaltenen Subelemente bei einem Inhaltsmodell

(*content model*). Dem Benutzer wird also ein pfadorientierter Zugang zu den als Eltern-Kind-Beziehungen angeordneten Knotenobjekten gewährt.

Betrachten wir das Beispiel aus Abbildung 1-2 als DOM-Baum, so finden wir eine direkte Analogie zu der für Menschen lesbaren Notation eben dieser Abbildung:

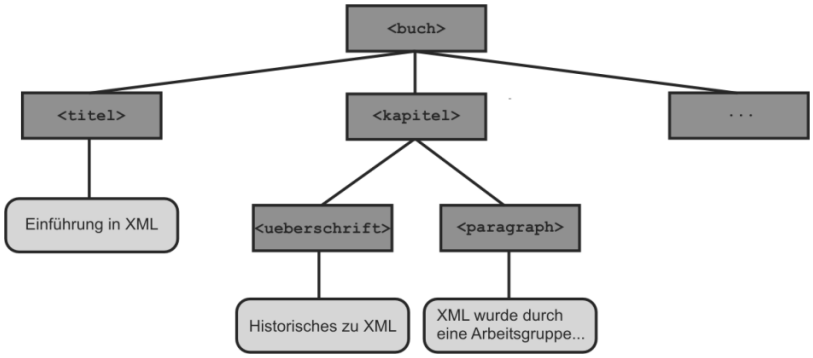


Abb. 1-12: *DOM-Baum des Dokuments aus Abb. 1-2*

Jeder dieser logischen Baumknoten wird durch ein DOM-Objekt repräsentiert, dessen übergeordnetes Objekt vom Typ *node* (Knoten) ist. Wir stellen die Definition dieses Basisobjekts ausschnittsweise in Abbildung 1-13 vor.

```
interface Node
{
  readonly attribute DOMString       nodeName;
  readonly attribute unsigned short  nodeType;
  readonly attribute Node            parentNode;
  readonly attribute NodeList        childNodes;
  readonly attribute NamedNodeMap    attributes;
  Node     insertBefore(in Node newChild, in Node refChild)
           raises(DOMException);
  Node     replaceChild(in Node newChild, Node oldChild)
           raises(DOMException);
}
```

Abb. 1-13: *Ausschnitt aus der DOM-Basisklasse »Node«*

Für die über *attribute* deklarierten Attribute dieser Objektklasse werden automatisch entsprechende get- und set-Methoden eingeführt, wie beispielsweise *set_parentNode(Node parentNode)*. Diese Methoden stehen somit zur Verfügung, wenngleich sie nicht explizit in der IDL-Notation aufgeführt sind.

Von der Basisklasse *Node* werden die spezialisierten Klassen *Document* zur Repräsentation der XML-Dokumentenwurzel, *Element* zur Repräsentation von XML-Elementen, *Attr* zur Repräsentation von XML-Attributen, *Entity* zum Zugriff auf XML-Einheiten usw. abgeleitet. Jede abgeleitete Klasse enthält spezielle Schnittstellen, die für den Zugriff auf das dadurch repräsentierte XML-Kon-

strukt notwendig sind. So bietet beispielsweise die Klasse *Element* den Zugriff auf *Attribut*-Knoten, die ihrerseits jeweils Attributname und -wert enthalten.

```
interface Element : Node
{
  readonly attribute DOMString       tagName;
  Attr     getAttributeNode(in DOMString name);
  Attr     setAttributeNode(in Attr newAttr)
       raises(DOMException);
  Attr     removeAttributeNode(in Attr oldAttr)
       raises(DOMException);
  boolean  hasAttribute(in DOMString name);
  NodeList getElementsByTagName(in DOMString name);
}

interface Attr : Node
{
  readonly attribute DOMString       name;
  readonly attribute boolean         specified;
           attribute DOMString       value;
  readonly attribute Element         ownerElement;
};
```

Abb. 1-14: *Ausschnitt aus DOM-Element und DOM-Attr*

Die Verarbeitung von XML-Dokumenten mittels DOM-Implementierungen entspricht der Durchmusterung und Manipulation attributierter Bäume: Ausgehend von dem Dokumentenwurzelknoten navigiert eine Anwendung durch den Baum und nutzt die Schnittstellen der jeweiligen Knoten zum Auslesen der damit korrespondierenden XML-Dokumentinhalte.

1.5.2 Simple API for XML (SAX)

Das *Simple API for XML* entstand aus der Überlegung heraus, dass es bisweilen unrentabel sein kann, einen Parserbaum zu erstellen und diesen zunächst in eine andere, interne Baumdarstellung zu überführen. Stattdessen sollte gleich beim Prozess des Parsens das Auftreten von Markierungen als Ereignis angesehen werden, die in der Regel einen Beginn und ein Ende haben. Damit wären auch etwa auftretende Speicherbeschränkungen beseitigt, die dazu führen könnten, dass große XML-Dokumente intern nicht mehr dargestellt werden könnten. Abgesehen davon, dass man dergleichen Beschränkungen sicher auch anders umgehen kann, hat sich das SAX als Alternative zu DOM mittlerweile etabliert – wenngleich nicht in demselben Umfang wie DOM und nicht standardisiert.

Eine SAX-Implementierung stellt einen Rahmen (*framework*) dar, dessen Interface ebenso wie DOM als Bibliothek vorliegt, um in eigene Programme eingebunden zu werden. Da SAX aber ereignisgesteuert arbeitet, ergibt sich ein völlig anderer Anwendungsaufbau, was die eigentliche Dokumentverarbeitung angeht. Während bei DOM das Anwendungsprogramm quasi die Führungsrolle innehat

und die DOM-API-Funktionen aufruft, mit denen der Parserbaum ausgewertet wird, erzeugt das SAX-API den Parserbaum selbst und übergibt bei Auftreten bestimmter Markierungen (z.B. Beginn der Markierung **<kapitel>** und Ende der Markierung **</kapitel>**) entsprechende Ereignisse an das Anwendungsprogramm, auf die es nach der Art von Expertensystemen reagieren muss und die SAX-Funktionen aufruft, die zur Verarbeitung der Elemente bzw. ihrer Inhalte notwendig sind.

Je nachdem ob die Anwendung für das jeweilige Ereignis eine Empfängermethode implementiert hat oder nicht, kann sie auf dieses Ereignis reagieren oder es einfach ignorieren. Reagiert sie auf ein Ereignis, so hält das SAX den Parserprozess bis zur vollständigen Abarbeitung des Ereignisses an. Anschließend fährt es fort und sendet bei Bedarf weitere Ereignisse.

Zum Abfangen dieser Ereignisse wird in objektorientierten SAX-Implementierungen eine Verfahrensweise angewendet, die u.a. auch aus der objektorientierten Programmierung von ereignisgesteuerten Benutzeroberflächen bekannt ist: Im SAX-Framework sind Klassen und Methoden zum Abfangen der Ereignisse definiert. Eine Anwendung leitet ihre jeweiligen Klassen von denen des Frameworks ab und überlädt die mit den Ereignissen korrespondierenden Methoden. Zur Ausführungszeit werden dann die zum Abfangen der Ereignisse vorgesehenen anwendungseigenen Methoden aufgerufen.

```
from xml.sax import Contenthandler, ...
from xml.sax import make_parser

##### Klasse CDocHandler
class CDocHandler(ContentHandler):
  def startElement(self, elementName, elementAttribute):
    # Handelt es sich um das Element „
    if elementName == "kapitel":
      # Auslesen des Attributs nr des Elements <kapitel>
      nr = attribute.get( "nr", None );
      print "Kapitel " + nr
…

##### Hauptprogramm
# Instanziierung der eigenen Klasse
myDocHandler = CDocHandler()

# Parserinstanz erstellen
parser = make_parser()

# Eigene Implementierung im Parser registrieren
parser.setContentHandler( myDocHandler )

# Parser-Prozess initiieren.
# Verwendung der Standardeingabe (Kommandozeile) als Datenquelle
parser.parse(sys.stdin)
```

Abb. 1-15: *Beispiel eines SAX-Frameworks zum Abfangen von Start-Tags*

Das Beispiel aus Abbildung 1-15 gibt einen Ausschnitt eines in der Programmiersprache Python geschriebenen Programms wieder, welches Dokumente der DTD *buch* (siehe unser Beispiel in den Abbildungen 1-2 und 1-4) verarbeitet. Hierzu erbt das Programm von der SAX-Klasse *ContentHandler* und überlädt die Methode *startElement*.

Im Hauptprogramm wird eine Instanz der eigenen Klasse *CDocHandler* erstellt und im Parser registriert. Dieser ruft bei jedem Auftreten der Startmarke eines Elements die überladene Methode *startElement* auf, d.h., er sendet das Ereignis *startElement*. Wie aus dem Beispiel ersichtlich, werden der Methode zudem der Name des betroffenen XML-Elements und eine Referenz auf die Attribute übergeben. Beides nutzen wir im Beispiel, um bei Auftreten des Elements *kapitel* das Attribut *nr* auf dem Bildschirm auszugeben.

Sofern für das zugrunde liegende XML-Dokument eine DTD existiert und auch verwendet wird, muss nicht überprüft werden, ob das Attribut *nr* auch angegeben wurde: SAX-Parser prüfen auf Wohlgeformtheit und bei Bedarf auch auf Gültigkeit. Treten Verletzungen bzgl. einer dieser Eigenschaften auf, so können Fehler wiederum durch dafür vorgesehene Ereignisse abgefangen werden.

1.5.3 Zusammenfassung

Zusammenfassend fragen wir uns, welche Vor- bzw. Nachteile SAX gegenüber DOM bringt? Ganz analog zur Diskussion im Compilerbau über Mehrphasencompiler oder Übersetzer mit rekursivem Abstieg sehen wir dabei folgende Eigenschaften:

- SAX verhindert eine mehrfache Repräsentation des XML-Dokuments: Im Unterschied zu DOM werden in SAX nur die für das jeweilige Ereignis notwendigen Strukturen temporär angelegt, während bei DOM der Parserbaum komplett als solcher vorliegt. Nach Abarbeitung eines Ereignisses werden die damit verbundenen Datenstrukturen zerstört. Es obliegt allerdings der benachrichtigten Anwendung, das gesamte Dokument oder Teile dessen im Speicher selbst abzubilden, d.h., SAX verbietet nicht die Erstellung eines Parserbaumes, es erzeugt lediglich selbst keinen vollständigen. Dies erspart grundsätzlich Transformationen zwischen verschiedenen, internen Repräsentationen.
- SAX bietet einen bedingt möglichen Eingriff in den Parser-Prozess: Dadurch, dass der Parserprozess und die Anwendung synchron laufen, kann Letztere in den Prozess eingreifen. Sinnvoll ist ein solcher Eingriff zum Beispiel über die Auswertung der in Abschnitt 1.4.3 eingeführten Verarbeitungsanweisungen (*processing instructions*).

Nachteilig kann sich auswirken , dass SAX keinen permanenten Zugriff auf den Parserbaum gewährt, sondern nur einen Ereignisstrom aus dem Auftreten von Markierungen generiert. Ist ein Ereignis abgearbeitet und sind im Anwendungsprogramm keine weiteren Vorbereitungen getroffen, den betreffenden Knoten

und seine Inhalte programmintern zu verwalten, hat die Anwendung keine Möglichkeit, nochmals auf diesen Knoten zuzugreifen. Jedes Ereignis wird nämlich nur einmal erzeugt und ist danach vergangen.

Der eingangs erwähnte Vorteil, dass SAX-Implementierungen speichereffizienter arbeiten als das DOM und dadurch wiederum insbesondere zur Verarbeitung sehr großer XML-Dokumente geeignet sind, muss ebenfalls in Frage gestellt werden, wenn die Speicherung von XML-Dokumenten und DOM-Bäumen in Datenbanken erfolgt. Diese sind bekanntlich für den wahlfreien Zugriff auf umfangreiche Datenbestände konzipiert und beheben damit den Nachteil des nicht wahlfreien Zugriffs auf eine Baumstruktur.

Die Verwendung des einen oder des anderen APIs ist daher wohl – wenn nicht spezielle Nebenbedingungen die freie Auswahl beschränken – eher eine Frage der eigenen persönlichen Philosophie im Sinne einer Affinität zu ereignisgesteuerter Programmierung oder navigierendem Zugriff.

1.6 Diskussion

Fassen wir das bislang Gesagte zusammen, so stellt sich XML als ein semistrukturiertes Datenmodell dar, dessen Eigenschaften für einen Einsatz insbesondere im Bereich des Internets geeignet erscheinem: Es ist ein einfaches, gut lesbares und in sich wohlstrukturiertes Modell, welches dem hoch interaktiven Charakter des Internets – im Sinne einer Mensch-Maschine-Interaktion – gerecht wird.

Wir haben gesehen, dass auch die rechnerinterne Darstellung und Verarbeitung von XML-Dokumenten vorgesehen ist und sich durch die Verwendung bzw. Übernahme von Standards aus anderen Bereichen (z.B. CORBA) eine hohe Kompatibilität zwischen unterschiedlichen Systemen ergibt – was ebenfalls für den Einsatz von XML im heterogenen Umfeld des Internets spricht.

Dennoch erscheint es angebracht zu überlegen, ob mit den bisher eingeführten Konzepten wirklich alle Notwendigkeiten für den letztendlichen Einsatz von XML als Austauschformat im Internet – und damit natürlich auch für Web-Datenbanken – abgedeckt werden können.

Austauschformate bedürfen stets einer gewissen Mächtigkeit und Universalität – was durchaus nicht immer mit dem Umfang an Konzepten gleichzusetzen ist! Man erkennt dies beispielsweise am objektorientierten Datenmodell, welches sehr häufig als unifizierendes Modell (z.T. auch als kanonisches Datenmodell bezeichnet) in heterogenen Umgebungen eingesetzt wird, oder auch an dem extrem allgemein gültigen Graphmodell, welches den semistrukturierten Datenmodellen als Grundlage dient.

Obwohl einfach in der Konzeption und mächtig in der Ausdrucksform, reicht ein Graphmodell allein zur Verwaltung bzw. zum Austausch von Web-Daten nicht aus. Für OEM ist z.B. erkannt worden, dass es für die Modellierung heterogener Daten sinnvoll ist, eine Anzahl vordefinierter Datentypen bereitzuhalten. Hier ist ein Nachteil von XML zu erkennen, denn genau genommen gibt es nur einen

Datentyp, den Datentyp STRING. So ist es bei der Deklaration eines Elementtyps in einer DTD nicht möglich anzugeben, dass dessen Inhalt numerisch oder ein Wahrheitswert sein soll. Wenngleich man diesen Umstand auch als Vereinfachung für den XML-Prozessor ansehen kann, ergeben sich dennoch berechtigte Fragen, wenn es um das Thema der Datenintegrität beim Austausch von Daten in Web-Datenbanken geht.

Ein weiteres Problem stellen Attribute dar. In relationalen Schemata bzw. in objektorientierten Klassen werden Attribute stets lokal bezüglich des Schemas oder der Klasse vereinbart. Somit kann beispielsweise ein Attribut mit dem Namen *bezeichner* in mehreren, nicht über die Vererbungshierarchie in Beziehung stehenden Klassen definiert werden. Elementtypen in XML werden hingegen stets global in einer DTD definiert. Auch wenn, wie in Abbildung 1-16 angedeutet, ein Element B ausschließlich in einem anderen, Nicht-Wurzelelementtyp A vorkommt, so ist es trotzdem global definiert. Das heißt, der Bezeichner B kann für keinen weiteren Elementtyp verwendet werden.

```
<!ELEMENT A (B)>
<!ELEMENT B (#PCDATA)>
```

Abb. 1-16: *Elementtypen werden stets global vereinbart*

Verhältnismäßig rudimentär ist in XML auch die Referenzierung interner und externer Elemente bzw. Dokumente. Sie reduziert sich überwiegend auf den Verweis auf den Ort einer zu referenzierenden Quelle; in physischen Einheiten kann zudem der Typ der Quelle angegeben werden. Oft ist es aber notwendig, eine Referenz mit weiteren Informationen (Metadaten) zu versehen, was zwar durch die Einführung entsprechender Elementtypen und Attribute möglich, jedoch nicht standardisiert ist. Auch ist kein einheitliches Verfahren zur Referenzierung mehrerer externer Quellen, ausgehend von einem Element, vorhanden. Jedes System nutzt somit seine eigene Notation zur Realisierung von 1:n-Beziehungen.

Mit den aufgezeigten, fehlenden Eigenschaften ist XML sicherlich nur bedingt zur Verwaltung und zum Austausch von Web-Daten geeignet. Sollen nämlich Systeme im World Wide Web interagieren, so verbietet die Heterogenität der partizipierenden Systeme solche Freiheitsgrade, wie eben bei der Angabe von Referenzen oder 1:n-Beziehungen möglich. Mit anderen Worten: Bestehende Freiheitsgrade müssen durch entsprechende Standards eingeschränkt oder zumindest im Konsens festgeschrieben werden.

Derzeit findet auch genau diese Standardisierung in Form der Ausarbeitungen für *XML Namespaces* [BrHL99], *XLink* [XLink01], *XPointer* und *XML-Schema* statt. Wir gehen auf diese Entwicklungen in unserem Kapitel nicht weiter ein, sondern verweisen einerseits auf die Literaturangaben und andererseits auf nachfolgende Kapitel dieses Buches. Es sei lediglich erwähnt, dass das Problem der globalen Namensdeklaration durch XML Namespaces, so genannte Namensräume für XML, behoben wird und die Konzepte zu XLink und XPointer die Referen-

zierungsmöglichkeiten in XML erweitern. Auch XML-Schemata werden an anderer Stelle in diesem Buch ausführlich behandelt (Kapitel 2), so dass wir hier nur kurz darauf hinweisen, dass sie mächtige Konstrukte zur Definition von Typen in XML einführen, die als wesentliche Erweiterung der Dokumenttypdefinition anzusehen sind. Interessant hieran ist, dass die Schemadefinitionen in XML selbst geschrieben werden, womit wir noch einmal an die selbstbeschreibende Eigenschaft eines semistrukturierten Datenmodells erinnern können.

1.7 Zusammenfassung

In diesem Kapitel haben wir aufgezeigt, dass semistrukturierte Datenmodelle dem Grundsatz allgemeiner Graphen folgen und zur Verwaltung von Daten im World Wide Web besonders geeignet sind. Wir haben aus diesem Typus von Modellen kurz das *Object Exchange Model* (OEM) als rechnerinternes Datenmodell und anschließend die *eXtensible Markup Language* (XML) als ein textbasiertes Austauschmodell vorgestellt.

Die Einführung in grundlegende Konzepte dieser Sprache bzw. dieses Datenmodells folgten, um dem Leser eine Basis zum Verständnis nachfolgender Kapitel dieses Buches zu geben.

Im Anschluss an die kurze Einführung haben wir den Zusammenhang zwischen einem externen und einem rechnerinternen Modell erläutert und zwei Alternativen vorgestellt, wie rechnerintern mit XML-Dokumenten gearbeitet werden kann.

Eine Diskussion über erkennbare Schwächen des reinen Datenmodells XML und die Hinweise auf derzeit aktuelle Arbeiten zu diesem Thema bildeten den Abschluss.

Literatur

[BrHL99] T. Bray, D. Hollander, A. Layman. *Namespaces in XML*. W3C-Konsortium, 1999, http//www.w3.org/TR/1999/REC-xml-names-19990114.

[CGHI94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. *The TSIMMIS Project: Integration of Heterogeneous Information Sources*. In Proceedings of IPSJ Conference, pp. 7-18, Tokyo, Japan, 1994.

[DOM02] A. Le Hors et al. *Document Object Model (DOM) Level 3 Core Specification*. W3C Working Draft, http://www.w3.org/TR/2002/WD-DOM-Level-3-Core-20020409, 2002.

[Ecks99] R.Eckstein. *XML Pocket Reference*. O'Reilly & Associates Inc., 1999.

[FFLS00] M. Fernandez, D. Florescu, A. Levy, D. Suciu. *Declarative Specification of Web Sites with Strudel*, In VLDB Journal (9)1, pp. 38-55, 2000.

[GoPr98] C.F. Goldfarb, P. Prescod. *The XML Handbook*. Prentice Hall, 1998.

[MAGQ97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: *A Database Management System for Semistructured Data*. In SIGMOD Record, 26(3), pp. 54-66, 1997.

[MAMM98], G. Mecca, P. Atzeni, A. Masci, P. Merialdo, G. Sindoni. *The Araneus Web-Based Management System*. In Exhibits Program of ACM SIGMOD, 1998.

[PaGW95] Y. Papakonstantinou, H. Gracia-Molina, J. Widom. *Object Exchange Across Heterogeneous Information Sources*. In Proceedings of the IEEE International Conference on Data Engineering, pp. 251-260, 1995.

[Schema01] D. C. Fallside. *XML Schema Part 0: Primer.* W3C-Konsortium, http://www.w3.org/TR/xmlschema-0/, 2001.

[Suci01] D. Suciu. *Typechecking for Semistructured Data*. In Proceedings of the International Workshop on Database Programming Languages, 2001.

[XLink01] S. DeRose, E. Maler, D. Orchard. *XML Linking Language (Xlink) Version 1.0*. W3C-Konsortium, http://www.w3.org/TR/2000/REC-xlink-20010627, 2001.

[XML00] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler. *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C-Konsortium, 2000, http://www.w3.org/TR/2000/REC-xml-20001006.

[XQuery02] M. Fernández, J. Marsh, M. Nagy. *XQuery 1.0 and XPath 2.0 Data Model*. W3C-Konsortium, 2002, http://www.w3.org/TR/query-datamodel/.

2 XML Schema

Harald Schöning , Walter Waterfeld

Kurzfassung

W3C XML Schema bietet im Gegensatz zu DTD mächtigere Modellierungskonzepte. Dazu gehören ein umfangreiches Typsystem, die Möglichkeit, lokale Definitionen vorzunehmen, ein Vererbungsmechanismus und Beschränkungen. XML Schema ermöglicht damit die Verwendung verschiedener Modellierungsstile. Diese Vielfalt wird mit einer sehr hohen Komplexität des W3C-XML-Schema-Standards erkauft. Die vielfältigen Ausdrucksmöglichkeiten in XML Schema bieten Raum für verschiedene Modellierungsstile, so dass inhaltlich äquivalente Informationsmodelle auf ganz unterschiedliche Weise formuliert werden können.

2.1 Einleitung

Wohlgeformte XML-Dokumente haben in vielen Anwendungen ihren Platz, auch wenn sie keine explizite schematische Information besitzen. Der XML-1.0-Standard[1] lässt solche schemalosen Dokumente ausdrücklich zu. Trotzdem besteht in wichtigen Anwendungsgebieten auch der Bedarf für schematische Beschreibung von XML-Dokumenten. Der XML-1.0-Standard hat mit dem DTD-Konzept einen Mechanismus dafür definiert, der manche Defizite aufweist. Daher haben sich mittlerweile mehrere Ansätze entwickelt, die DTD als Mittel der Schemadefinition ablösen sollen. Einer dieser Ansätze ist der W3C-XML-Schema-Standard.

In diesem Kapitel wird zunächst aufgezeigt, welche Rolle Schemata in XML im Gegensatz zu den aus Datenbanken bekannten Schemata spielen. Nach einem kurzen Überblick über andere Schemadefinitionssprachen für XML wird der W3C-XML-Schema-Standard ausführlich vorgestellt. Damit ist ein Vergleich von XML Schema mit dem in Datenbanken üblichen relationalen Datenmodell möglich. Die mit W3C XML Schema möglichen Modellierungsstile von XML-Schemata werden anschließend diskutiert. Ein kurzer Überblick über die Möglichkeiten zur Definition des physischen Schemas schließt das Kapitel ab.

1. W3C recommendation

2.2 Rolle von Schemata

Die Rolle von Schemata in Datenbanken und für XML-Instanzen weist starke Unterschiede auf. Hier wollen wir zunächst nur die prinzipiellen Unterschiede diskutieren. Die Datenmodell-Unterschiede zwischen konventionellen relationalen Datenbanksystemen (RDBMS) und XML-Schema diskutieren wir in Kapitel 2.5 nach der Vorstellung von XML Schema.

2.2.1 Schemata in Datenbanken

Die Unterscheidung zwischen Metadaten und Daten beziehungsweise zwischen Schemata und Instanzen ist ein wichtiges Konzept in Datenbanken. Ein Schema beschreibt die Struktur der Instanzen. Es ist Voraussetzung zur Speicherung und Verwaltung von Instanzen. Die Instanzen können nicht ohne Schema existieren. Die Realisierung der Verwaltungsfunktionen ist darauf ausgerichtet, dass das Schema im Vergleich zu den Instanzen wesentlich weniger Änderungen unterworfen ist.

Im Schema kann man zwischen Typdefinitionen und Integritätsbedingungen unterscheiden. Der Typ kann zur Compilezeit der Anwendungen überprüft werden, während für die Prüfungen der Integritätsbedingungen Laufzeitprüfungen erforderlich sind.

Neben den Typdefinitionen und Integritätsbedingungen sind für ein Datenbankschema auch die Definition von Behältern oder Einstiegspunkte für Instanzen wichtig.

Die bisher genannten Aspekte betreffen das logische Schema, da sie das Verhalten von Anwendungsprogrammen beeinflussen. Daneben spielt in Datenbanken das physische Schema eine wichtige Rolle. Damit ist es möglich, die Art der Speicherung und Indexierung zu beeinflussen. Durch die Definition dieser Eigenschaften auf einer speziellen Schemaebene bleiben Anwendungsprogramme unabhängig von diesen physischen Eigenschaften.

2.2.2 Schemata für XML

Im Gegensatz zu Daten in konventionellen Datenbanken sind XML-Instanzen selbstbeschreibend. Sie enthalten neben den Werten in jeder Instanz die so genannten »Tags«. Generell ist damit eine Interpretation ohne Schema möglich. Allerdings enthalten die meisten XML-Schemasprachen auch Funktionalität wie vorbelegte Werte und Konstanten, welche die in der Instanz enthaltenen Informationen ergänzen.

Für XML bestimmt die Instanz selber, ob und welches Schema sie hat. Es ist sogar möglich, verschiedenen Teilen einer Instanz unterschiedliche Schemata zuzuordnen. Diese Offenheit steht im starken Gegensatz zu Datenbanken, wo die Instanzen durch Schemata disjunkt klassifiziert werden.

Die optionalen Schemata in XML entsprechen damit eher der Rolle von Integritätsbedingungen in Datenbanksystemen. Die Überprüfung erfolgt durch die so genannte Validierung – üblicherweise beim Parsen der XML-Instanz. Dies gilt in jedem Fall für die DTDs als erste XML-Schemasprache. Der aktuelle W3C-XML-Schema-Standard enthält jedoch inzwischen auch ausgefeilte Typdefinitionen, die durch ihre statische Prüfbarkeit in Anwendungsprogrammen über Integritätsbedingungen hinausgehen. Im Gegensatz zu Datenbanksystemen betreffen die in den XML-Schema-Standards formulierbaren Integritätsbedingungen jedoch nur eine Instanz.

2.3 XML-Schemasprachen: Überblick und Entwicklung

2.3.1 DTD

Die erste und immer noch am weitesten verbreitete Schemasprache für XML ist DTD (*Document Type Definition*). Ihre Definition ist integraler Bestandteil des ersten XML-1.0-Standards [BPSM00]. Damit ist sie eng in die Definition für XML-Instanzen integriert. Sie stellt wie XML insgesamt eine vereinfachte Form eines SGML-Bestandteils dar – in diesem Fall der gleichnamigen SGML-DTDs.

Die Hauptfähigkeit von DTDs besteht in der Beschreibung von Inhaltsmodellen, die in allen Schemasprachen vorhanden ist. Inhaltsmodelle definieren über eine eigene Modellierungssprache die innerhalb eines Elements erlaubten Elemente. Im folgenden Beispiel wird definiert, dass ein *Telefonkontakt* mehrere *Vornamen*, einen *Nachnamen*, eine *Telefonnummer* und eine optionale *Faxnummer* hat.

```
<!ELEMENT TelefonKontakt (Vorname*, Nachname, Telefonnummer, Faxnummer?) >
<!ELEMENT Vorname (#PCDATA) >
<!ELEMENT Nachname (#PCDATA) >
<!ELEMENT Telefonnummer (#PCDATA) >
<!ELEMENT Faxnummer (#PCDATA) >
```

Zusätzlich besitzen DTDs mit dem »*General Entitiy*« eine spezielle Fähigkeit, die man nicht den üblichen Schemafähigkeiten zuordnen kann. Ein »*General Entitiy*« ist eine Art Makro, das ein beliebiges XML-Fragment definiert. Dieses Makro kann nun an beliebigen Stellen in der XML-Instanz verwendet werden.

Auch aufgrund dieses dokumentenorientierten Erbes haben DTDs als Schemata eine Reihe von Nachteilen:

- Kein XML. Eine der gravierendsten Probleme insbesondere für die Weiterverarbeitung in Werkzeugen ist, dass DTDs keine XML-Syntax haben.
- Keine Datentypen. Aufgrund ihrer dokumentenorientierten Herkunft kennen DTDs im Wesentlichen nur den Datentyp *String*. Es gibt nur einige wenige zusätzliche Datentypen wie *Token* und *Liste von Tokens*. Diese können dann

aber nur für Attribute benutzt werden. Dies schränkt die Nutzung von DTDs für datenorientierte Anwendungen gravierend ein.
- Keine Namensräume. XML-Namensräume [BrHL99] haben sich inzwischen als weitere XML-Norm weitgehend durchgesetzt. DTDs für Instanzen mit XML-Namensräumen lassen sich nur mit »Tricks« definieren, die die Benutzbarkeit der DTD als Schema stark einschränken.

2.3.2 Andere XML-basierte Schemasprachen

Aufgrund der Defizite von DTDs kam es sehr schnell zu der Entwicklung einer Vielzahl von Schemasprachen. Eine weitere Ursache war die schleppend verlaufende Standardisierung von XML Schema. Obwohl diese Schemasprachen unterschiedliche Ansätze verfolgten, ist ihnen doch die Formulierung in XML, die Unterstützung von XML-Namensräumen und die Verarbeitung von primitiven Datentypen gemeinsam. Beispiele für solche Schemasprachen sind [LC00]:

- XDR (*XML-Data Reduced*). Eine Schemasprache, die noch häufig in Microsoft-XML-Infrastrukturen benutzt wird
- SOX (*Schema for Object-Oriented XML*). Eine Schemasprache, die sich an Modellierungskonzepte in objektorientierten Programmiersprachen anlehnt.
- *Schematron*. Eine regelbasierte Schemasprache zur Definition von ausgefeilten Integritätsbedingungen.

2.3.3 XML Schema

Aufgrund der Vielzahl entstehender Schemasprachen wurde der Bedarf für eine Standardisierung klar ersichtlich [Wa01]. Ein Ziel war, sowohl die dokumentenorientierte als auch die datenorientierte Modellierung zu unterstützen. Eines der wichtigsten Merkmale ist die starke Unterstützung von einfachen und komplexen Typen. Man ging aber noch weiter und versuchte, fast alle Modellierungsstile der vorhandenen Schemasprachen in der Standard-XML-Schemasprache zu integrieren. Das beinhaltete ausgefeilte Vererbungsmechanismen zwischen benutzerdefinierten komplexen Typen, die aus der objektorientierten Modellierung übernommen wurden. Hier wurden natürlich nur die strukturellen Aspekte berücksichtigt. XML Schema enthält keinerlei Verhaltensbeschreibung.

Neben objektorientierten Konzepten sind auch die Fähigkeiten von DTDs vollständig in anderer Syntax in XML Schema enthalten. Die einzige Ausnahme sind »*General Entities*«. So entspricht das in Abschnitt 2.3.1 angegebene DTD-Fragment genau dem folgenden XML-Schema-Fragment:

```
<xs:element name="TelefonKontakt">
  <complexType>
    <xs:sequence>
      <xs:element ref="Vorname" maxOccurs="unbounded"/>
      <xs:element ref="Nachname"/>
      <xs:element ref="Telefonnummer"/>
      <xs:element ref="Faxnummer" minOccurs="0"/>
    </xs:sequence>
  </complexType>
</xs:element>
<xs:element name="Vorname" type="xs:string"/>
<xs:element name="Nachname" type="xs:string"/>
<xs:element name="Telefonnummer" type="xs:string"/>
<xs:element name="Faxnummer" type="xs:string"/>
```

Auch die Inhaltsmodelle von Elementen als wichtigste DTD-Modellierungsfähigkeit ist bei XML Schema erweitert wurden, beispielsweise um beliebige Kardinalitäten und in eingeschränkter Weise um die Definition von ungeordneten Mengen von Elementen.

Die Unterstützung der Fähigkeiten von vielen Schemasprachen ist einer der Gründe, warum XML Schema relativ komplex geworden ist.

2.3.4 Relax NG

Durch W3C XML Schema wurden die Nachteile von DTDs beseitigt. Der Standard hat auch genügend Vorteile, um die meisten anderen zwischenzeitlich entstandenen Schemasprachen ablösen zu können.

Ein wesentliches Manko des Standards ist jedoch dessen Komplexität. Daher werden weiterhin Schemasprachen entwickelt, die wesentlich einfacher sind. Ein wichtiges Beispiel dafür ist *Relax* [CM01], das inzwischen in einer verbesserten Form *Relax NG* genannt wird.

Die Einfachheit wird vor allem dadurch erreicht, dass man sich nur auf Integritätsbedingungen beschränkt. Im Gegensatz zu XML Schema wird der Informationsgehalt der Instanz also nicht um Default-Werte oder Typinformation aus einem Schema angereichert. Dabei bietet Relax NG sogar etwas mächtigere Inhaltsmodelle als XML Schema an, beispielsweise gibt es im Gegensatz zu XML Schema kaum Einschränkungen bei der Definition von ungeordneten Mengen von Elementen. Eine weitere starke Vereinfachung gegenüber XML Schema wurde dadurch erreicht, dass man auf sämtliche benutzerdefinierten Typkonzepte und insbesondere auf Typvererbung verzichtete. Man kann sich jedoch fragen, ob für das Hauptziel der Reduzierung der Komplexität nicht statt einer neuen Schemasprache eine Teilmengenbildung von XML Schema ausreichend wäre.

2.4 W3C XML Schema

Die *XML Schema Recommendation* des World Wide Web Consortium besteht aus drei Teilen: Teil 2 [BiMa01], der Datentypdefinitionen behandelt, Teil 1 [TBMM01], der andere Schemastrukturen spezifiziert, und Teil 0 [Fall01], dem so genannten »Primer«, der keinen Standard festlegt, sondern lediglich eine Erläuterung für die Teile 1 und 2 darstellt.

Das Ziel von XML Schema ist, eine Typdefinition für XML-Dokumente bereitzustellen, gegen die diese Dokumente validiert werden können, und gleichzeitig Information explizit zu machen, die in den Dokumenten ggf. implizit ist, wie z.B. Normalisierungsanforderungen an Werte oder Vorbelegungswerte (default values). In diesen beiden Aspekten kann XML Schema die DTD aus XML 1.0 [BPSM00] ablösen, womit viele bei DTD beklagte Defizite z. B. bezüglich der Modellierungsmächtigkeit entfallen.

Ein grundlegendes Konzept von XML Schema ist, Typen zu definieren und dann über Deklarationen Namen zu Typen zuzuordnen, um so Beschränkungen für das Auftreten von Elementen und Attributen in entsprechenden XML-Dokumenten zu spezifizieren. Wir werden daher zunächst auf das Typkonzept von XML Schema eingehen und dann auf die in XML Schema möglichen Definitionen. Darauf aufbauend werden wir die Möglichkeiten zur Deklaration behandeln und anschließend weitere Konzepte von XML Schema einführen.

2.4.1 Typkonzept

XML-Schema hat ein erweiterbares Typkonzept: Eine Reihe von Datentypen sind vordefiniert. Auf dieser Basis können neue Typen definiert werden, wobei zwischen einfachen und komplexen Typen unterschieden wird.

Vordefinierte Datentypen

In XML Schema sind einfache Datentypen vordefiniert. Die folgende Abbildung 2-1 [BiMa01] gibt einen Überblick über diese Typen und wie sie voneinander abgeleitet sind.

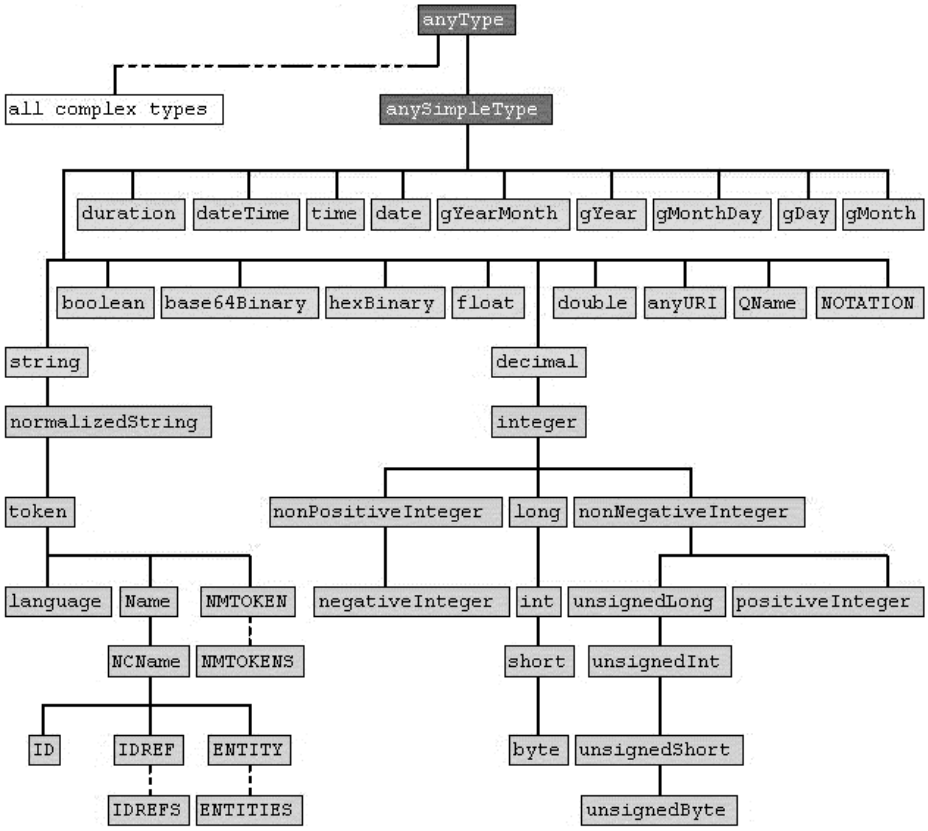


Abb. 2-1: *Datentypen von XML Schema*

Eine durchgezogene Linie steht dabei für Ableitung durch Einschränkung, eine gestrichtelte Linie für Ableitung durch Listenbildung (eine Erklärung dieser Ableitungsmechanismen findet sich unten). *anyType* und *anySimpleType* sind so genannte Ur-Typen. Jeder Typ ist von anyType abgeleitet, jeder einfache Typ ist von anySimpleType abgeleitet. Komplexe Typen sind von anyType durch eine Kombination von Einschränkungen und Erweiterungen abgeleitet.

Einfache und komplexe Typen

Ein *einfacher Typ* (*simple type*) im Sinne von XML Schema ist ein Datentyp, der weder Attribute noch Kindelemente beinhaltet – informell gesprochen also ein Datentyp, der geeignet ist, Attributwerte und den Inhalt von Elementen zu beschreiben, die nur textuellen Inhalt haben. Daher sind alle vordefinierten Datentypen einfache Typen.

Ein *komplexer Typ* (*complex type*) hingegen besteht aus einer (ggf. leeren) Menge von Attributdeklarationen und einem Inhaltsmodell. Dieses Inhaltsmodell kann Kindelemente fordern oder verbieten, textuellen Inhalt eines bestimmten

einfachen Typs zulassen und auch eine Mischung aus Elementen und textuellem Inhalt vorsehen (mixed content).

Komponenten einfacher Typen

Ein einfacher Datentyp in XML Schema besteht laut XML Schema Recommendation aus drei Komponenten

1. Eine Menge voneinander verschiedener Werte (*Wertebereich*, *value space*).
2. Eine Menge von Repräsentationen (*Repräsentationsraum*, *lexical space*).
3. Eine Menge von *Aspekten* (*facets*), die weitere Eigenschaften des Datentyps spezifizieren.

Leider sind diese drei Komponenten nicht orthogonal: Die beschränkenden Aspekte beeinflussen direkt den Wertebereich oder Repräsentationsraum.

Der Wertebereich

Der Wertebereich enthält alle Werte, die ein Datum des entsprechenden Datentyps annehmen kann. Einige dieser Werte können durch die Aspekte jedoch ausgeschlossen werden. Zu jedem Wert aus dem Wertebereich gibt es mindestens ein Literal im Repräsentationsraum.

Der Repräsentationsraum

Für die meisten Datentypen gilt, dass es für jeden Wert des Wertebereiches genau einen entsprechenden Wert des Repräsentationsraums (ein Literal) gibt. Für manche Typen, wie z.B. Zeittypen, ist es jedoch sinnvoll, verschiedene Literale zu haben, die demselben Wert entsprechen (verschiedene Zeitzonenangaben). Das gilt auch für den Datentyp float, bei dem die Literale 100 und 1.0E2 denselben Wert des Wertebereiches bezeichnen. Für die in XML Schema vordefinierten Datentypen ist in dem Fall, dass es mehrere Literale für denselben Wert gibt, eine kanonische Repräsentation definiert. Leider kann den vordefinierten Repräsentationen eines Wertes keine selbst definierte hinzugefügt werden.

Die Aspekte

Es gibt fundamentale Aspekte (Eigenschaften) von Datentypen, die durch Definition nicht beeinflusst werden können, und beschränkende Aspekte, die zur Einschränkung des Wertebereiches oder des Repräsentationsraumes benutzt werden können.

Zu den fundamentalen Aspekten zählen etwa:

- Die Eigenschaft der Ordnung eines Datentyps (voll, partiell, nicht geordnet). Der Datentyp string ist übrigens nicht geordnet. Das liegt daran, dass XML Schema in seiner jetzigen Fassung die Frage der sprachspezifischen Sortierung (collation) außer Acht lässt.
- Die Beschränktheit der Werte durch Unter- und/oder Obergrenze.

- Die Kardinalität. Hier wird unterschieden zwischen endlich und abzählbar unendlich.
- Die Tatsache, dass ein Datentyp numerisch ist.

Die beschränkenden Aspekte des Wertebereiches sind:

- Direkte Unter-/Obergrenzen für einen geordneten Wertereich: `minInclusive`, `minExclusive`, `maxInclusive`, `maxExclusive`,
- Längenbeschränkungen für Zeichenketten (aus Zeichen oder Hexadezimalzeichen), Namen, URI: `length` legt die genaue Länge fest. Alternativ kann eine Minimal- und/oder Maximallänge über `minLength` bzw. `maxLength` definiert werden.
- Längenbeschränkungen für Listen. Dort beschränken `length`, `minLength` und `maxLength` die Anzahl der zulässigen Listenelemente entsprechend.
- Längenbeschränkungen für Datentypen, die von decimal abgeleitet sind: `totalDigits` und `fractionDigits` beschränken die Gesamtzahl von Dezimalstellen und die Anzahl der Nachkommastellen. Trotz des irreführenden Namens legt `totalDigits` nicht die genaue Stellenzahl fest, sondern die maximale Stellenzahl.
- Aufzählung der erlaubten Werte (`enumeration`).

Der Repräsentationsraum kann eingeschränkt werden durch:

- Die Angabe eines regulären Ausdrucks, dem die Repräsentation genügen muss (`pattern`). Damit wird indirekt möglicherweise auch der Wertebereich eingeschränkt, nämlich dann, wenn alle Repräsentationen eines Wertes ausgeschlossen werden.

Ein bisschen aus der Rolle fällt der Aspekt `whiteSpace`. Er bestimmt, wie eine Zeichenkette zu behandeln ist, bevor die Validierung bezüglich des Datentyps vorgenommen wird. Anders formuliert beeinflusst dieser Aspekt, auf welchen Wert des Wertebereiches eine Repräsentation aus dem Repräsentationsraum abgebildet wird: Hat der Aspekt den Wert *preserve*, entspricht der Wert der Repräsentation. Bei *replace* werden alle Tabulator- und Zeilenende/Wagenrücklauf-Zeichen aus der Repräsentation durch Leerzeichen ersetzt, bei *collapse* außerdem noch führende und abschließende Leerzeichen unterdrückt und Folgen von Leerzeichen auf ein einziges reduziert.

In der heutigen Fassung erlaubt XML Schema nicht, eigene, selbst definierte Aspekte hinzuzufügen.

Erzeugung von Datentypen

Aus den in XML Schema vordefinierten Datentypen können neue Datentypen erzeugt werden. Auch die in XML Schema vordefinierten Datentypen sind zum Teil durch Ableitung aus primitiven Datentypen erzeugt. Die Wertebereiche der primitiven Datentypen (z.B. *float* oder *decimal*) sind axiomatisch festgelegt, der Wertebereich aller anderen Datentypen ergibt sich durch einen der unten ange-

führten Ableitungsmechanismen aus diesen Wertebereichen. Es gibt vier verschiedene Mechanismen, um neue Datentypen zu erzeugen. Die wesentlichen Mechanismen sind zwei verschiedene Vererbungsmechanismen:

- Einschränkung (*restriction*). Ein Typ kann aus seinem Basistyp durch Angabe von (null oder mehr) beschränkenden Aspekten abgeleitet werden. Wird kein beschränkender Aspekt angegeben, entspricht dies offensichtlich der Erzeugung eines gleichen Typs mit neuem Namen. Der neue Datentyp übernimmt Wertebereich und Repräsentationsraum, ggf. eingeschränkt durch die neuen beschränkenden Aspekte.
- Erweiterung (extension). Diese erzeugt immer einen komplexen Typ.

Neben diesen Vererbungsmechanismen gibt es noch zwei sehr spezielle Aggregationsmechanismen. Erstaunlicherweise erzeugen diese nur einfache Typen aus ebenfalls einfachen Typen:

- Listenbildung (*list*). Aus einem einfachen Typ kann ein einfacher Typ Liste gebildet werden.
- Vereinigung (*union*). Die Wertebereiche und Repräsentationsräume der einfachen Basistypen werden vereinigt.

2.4.2 Definitionen

XML Schema unterscheidet *Definitionen* (diese stellen interne Schemabausteine zur Verfügung, die ggf. wiederverwendet werden können) und *Deklarationen* (Information, die zur Validierung eines Dokumentes berücksichtigt wird).

Definition einfacher Typen

Wie bereits erwähnt, können einfache Typen durch Einschränkung, Listenbildung oder Vereinigung anderer einfacher Typen entstehen. Das erste Beispiel zeigt die Definition von Wassertemperatur auf Basis des vordefinierten Typs *number* [TBMM01]. Das Präfix *xs:* steht dabei für den Namensraum von XML Schema (s. Abschnitt Namensräume). Die Temperatur soll zwischen 0 und 100 Grad liegen und auf höchstens zwei Nachkommastellen genau sein:

```
<xs:simpleType name="Wassertemperatur">
   <xs:restriction base="xs:number">
      <xs:fractionDigits value="2"/>
      <xs:minInclusive value="0.00"/>
      <xs:maxInclusive value="100.00"/>
   </xs:restriction>
</xs:simpleType>
```

Das zweite Beispiel zeigt die Verwendung des Aspekts *pattern* bei der Definition einer deutschen Telefonnummer mit Vorwahl:

```
<xs:simpleType name="Telefonnummer">
   <xs:restriction base="xs:string">
      <xs:pattern value="0[0-9]+/[0-9]+"/>
   </xs:restriction/>
</xs:simpleType>
```

Eine Liste von Telefonnummern lässt sich dann definieren durch:

```
<xs:simpleType name="Telefonnummern">
   <xs:list itemType="Telefonnummer"/>
</xs:simpleType>
```

Wenn entsprechend zu *Telefonnummer* ein Datentyp *eMail* definiert ist, kann man den Datentyp *Kontakt* so definieren:

```
<xs:simpleType name="Kontakt">
   <xs:union memberTypes="Telefonnummer eMail"/>
</xs:simpleType>
```

Alle diese beispielhaften Typdefinitionen haben gemeinsam, dass ein benannter Typ definiert wurde. Es können jedoch auch anonyme Typen definiert werden (innerhalb von Deklarationen), die dann allerdings nicht wiederverwendet werden können.

Definition komplexer Typen

Komplexe Typen werden benötigt, wenn modelliert werden soll, dass Elemente Attribute haben, dass sie Kindelemente haben oder einen leeren Inhalt.

Komplexe Typen können einen einfachen Inhalt haben (d.h. keine Kindelemente, aber Inhalt) oder einen komplexen Inhalt. Der einfache Inhalt kann entweder durch Erweiterung oder durch Einschränkung eines Typs definiert werden. Das folgende Beispiel definiert einen komplexen Typ mit einfachem Inhalt, der benutzt werden kann, um eine Kontaktinformation zu speichern:

```
<xs:complexType name="KontaktInfo">
   <xs:simpleContent>
      <xs:extension base="Kontakt">
         <xs:attribute name="Name" type="xs:string"/>
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>
```

Der Inhalt eines mit diesem Typ definierten Elementes entspricht dem einfachen Typ *Kontakt*, wie er oben definiert wurde. Das Element hat außerdem ein Attribut *Name* vom Typ *string*. Folgendes Element entspricht dieser Typdefinition. Dabei wird deutlich, dass der Name des Elementes unabhängig von seinem Typ ist – der Typname tritt in einer entsprechenden XML-Instanz nicht auf. Der

Inhalt des Elementes ist einfach, enthält also keine Unterelemente. Trotzdem handelt es sich um einen komplexen Typ, da das Element ein Attribut besitzt.

```
<Kontakt Name="Eva Müller">09999/999999</Kontakt>
```

Komplexe Typen mit komplexem Inhalt erlauben die Angabe von Inhaltsmodellen. Es gibt drei verschiedene Inhaltsmodelle (in XML Schema auch Modellgruppen – *model groups* – genannt):

- Die Reihung (*sequence*), bei der die angeführten Einzelteile in der spezifizierten Reihenfolge vorkommen müssen
- Die Auswahl (Disjunktion, *choice*), bei der von den angeführten Einzelteilen nur eines auftreten darf
- Die Konjunktionsgruppe (*all*). Hier dürfen alle aufgeführten Teile maximal einmal auftreten, wobei deren Reihenfolge beliebig ist. Diese Gruppe hat Einschränkungen bezüglich der Verschachtelung, während eine Reihung eine Auswahl oder eine weitere Reihung enthalten darf, genau wie eine Auswahl eine weitere Reihung oder Auswahl umfassen darf.

Auch ein komplexer Typ mit komplexem Inhalt ist formal immer eine Erweiterung oder Einschränkung eines anderen Typs. Ist kein geeigneter anderer Typ vorhanden, kann auf den in XML Schema vordefinierten Ur-Typ *anyType* zurückgegriffen werden. Das folgende Beispiel demonstriert eine andere Modellierung einer Kontaktinformation, bestehend aus einem *Name*-Element gefolgt entweder von einem *Telefonnummer*-Element oder einem *eMail*-Element:

```
<xs:complexType name="Kontaktinformation">
   <xs:complexContent>
      <xs:restriction base="xs:anyType">
         <xs:sequence>
            <xs:element name="Name" type="xs:string"/>
            <xs:choice>
               <xs:element name="Telefonnummer"
                           type="Telefonnummer"/>
               <xs:element name="eMail" type="eMail"/>
            </xs:choice>
         </xs:sequence>
      </xs:restriction>
   </xs:complexContent>
</xs:complexType>
```

Da diese Einschränkung letztlich wenig aussagt, das Schema jedoch aufbläht, ist in diesem Fall eine verkürzte Schreibweise möglich:

```
<xs:complexType name="Kontaktinformation">
   <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:choice>
         <xs:element name="Telefonnummer"
                     type="Telefonnummer"/>
         <xs:element name="eMail" type="eMail"/>
       </xs:choice>
    </xs:sequence>
</xs:complexType>
```

Folgendes Beispielelement genügt dieser Typdefinition:

```
    <Kontakt><Name>Eva Müller</Name>
    <Telefonnummer>09999/999999</Telefonnummer></Kontakt>
```

Ein leerer Elementinhalt wird analog spezifiziert. Es handelt sich um einen komplexen Inhalt, da ja kein textueller Inhalt erlaubt sein soll. Dieser kann als Einschränkung von anyType definiert werden, wobei nur Attribute angegeben werden, also z. B.

```
<xs:complexType name="Kontaktart">
   <xs:complexContent>
      <xs:restriction base="xs:anyType">
         <xs:attribute name="Typ">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="eMail"/>
                  <xs:enumeration value="Telefon"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="Wert" type="Kontakt"/>
      </xs:restriction>
   </xs:complexContent>
</xs:complexType>
```

oder kurz:

```
<xs:complexType name="Kontaktart">
   <xs:attribute name="Typ">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="eMail"/>
            <xs:enumeration value="Telefon"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="Wert" type="Kontakt"/>
</xs:complexType>
```

Auch hier wieder ein Beispiel für ein Element, das dieser Typdefinition entspricht:

```
<Kontaktart Typ="Telefon" Wert="09999/999999"/>
```

Gemischter Inhalt (mixed content) wird durch Hinzufügen von `mixed="true"` als Attribut von *complexContent* spezifiziert.

Zur einfacheren Erstellung von Schemateilen und als eine Form der Unterstützung von Wiederverwendung gibt es noch zwei Hilfskonstrukte, die in XML Schema definiert werden können und in etwa mit den parameter entities der DTD verglichen werden können: die Modellgruppen und die Attributgruppen. Diese stellen benannte Gruppen von Elementen bzw. Attributen dar, die über eine Referenz einfach in einen Typ eingebunden werden können, wie das folgende Beispiel zeigt:

```
<xs:group name="Kgruppe">
  <xs:choice>
    <xs:element name="Telefonnr" type="Telefonnummer"/>
    <xs:element name="eMail" type="eMail"/>
  </xs:choice>
</xs:group>
<xs:attributeGroup name="Agruppe">
  <xs:attribute name="Name" type="xs:string"/>
  <xs:attribute name="Anrede" type="xs:string"/>
</xs:attributeGroup>
<xs:complexType name="Kontaktinformation">
  <xs:group ref="Kgruppe"/>
  <xs:attributeGroup ref="Agruppe"/>
</xs:complexType>
```

Der komplexe Typ wird nur über Referenzen auf die Gruppen definiert. Dies ist aber äquivalent dazu, die entsprechende Information in den komplexen Typ zu kopieren. Auch hier wieder ein Beispiel:

```
<Kontakt Name="Eva Müller" Anrede="Frau">
   <Telefonnr>09999/999999</Telefonnr>
</Kontakt>
```

2.4.3 Deklarationen

In Deklarationen werden Namen mit Typen, ggf. Vorbelegungswerten und Beschränkungen assoziiert. In XML Schema können Attribute, Elemente und Notationen deklariert werden. Diese Deklarationen können im Schema auf oberster Ebene stehen. In diesem Fall sind sie global und können von allen Stellen des Schemas aus referenziert werden. Sie können aber auch in andere Deklarationen oder in Definitionen verschachtelt sein. In diesem Fall sind sie lokaIndexierungl und nur in dem jeweiligen Kontext sichtbar. Um das zu erreichen, definiert XML Schema, dass Definitionen komplexer Typen ihren eigenen Gültigkeitsbereich für Element- und Attributnamen aufspannen. Solche Deklarationen waren in den Bei-

spieldefinitionen des vorigen Abschnittes schon enthalten. Folgendes Beispiel illustriert lokale und globale Deklarationen:

```
<xs:element name="Betrag">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name="Währung" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="Rechnung">
  <xs:sequence>
    <xs:element ref="Betrag"/>
    <xs:element name="Euro">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Betrag" type="xs:decimal"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:element>
```

Betrag als Kind von Rechnung bezieht sich auf das global definierte Element *Betrag*, während sich *Betrag* als Kind von *Euro* auf die lokale Definition (ohne das Attribut *Währung*) bezieht.

Wie die Beispiele illustrieren, kann man Typangaben einem Element oder Attribut zuordnen, indem man über das *type*-Attribut auf eine nicht anonyme Typdefinition verweist, oder indem man eine Typdefinition direkt in die Deklaration einbettet.

```
<xs:element name="Kontakt" type="Kontaktinformation"/>
```

Elemente können einen komplexen oder einfachen Typ haben. Bei Elementdeklarationen kann die Häufigkeit des Auftretens über die Attribute *minOccurs* und *maxOccurs* beschränkt werden, die jede nicht negative ganze Zahl annehmen können und mit 1 vorbelegt sind. Ferner gibt es die Attribute `default` und `fixed`, um Vorbelegungswerte oder feste Werte zu spezifizieren.

Attribute können nur einen einfachen Typ haben. Bei Attributdeklarationen kann über `use="optional"` bzw. `use="required"` der Effekt von #IMPLIED bzw. #REQUIRED der DTD erreicht werden. Auch hier gibt es die Attribute *default* und *fixed*.

Schließlich können auch Notationen in XML Schema deklariert werden. Ein Attributwert vom Typ NOTATION ist nur dann gültig, wenn die entsprechende Notation deklariert wurde. Beispiel:

```
<xs:notation
   name="jpeg" public="image/jpeg" system="viewer.exe"/>
```

Notationsdeklarationen können nicht lokal sein.

2.4.4 Offene Schemata

In manchen Anwendungen ist der genaue Aufbau von manchen Dokumentteilen nicht relevant oder ggf. nicht bekannt. Beispiel hierfür ist die Einbettung von XHTML in ein XML-Dokument. Die genaue Struktur des XHTML-Textes ist für die Schemadefinition irrelevant, interessant ist nur, dass es sich um XHTML handelt. Ein anderes Beispiel ist der Austausch von XML-Dokumenten zwischen verschiedenen Firmen. Eine Firma kann dabei das gemeinsam vereinbarte Schema erweitern (z.B. neue Attribute hinzufügen). Solange die benötigte Information noch vorhanden ist, kann die andere Firma dies getrost ignorieren.

XML Schema sieht für solche Zwecke mehrere Arten von Platzhaltern (*wildcards*) vor.

Als Platzhalter für Elemente gibt es any. Die erlaubten Elemente können nach ihrem Namensraum eingeschränkt sein: Entweder kann gefordert werden, dass sie bestimmten Namensräumen angehören, dass sie unqualifiziert sind oder dass sie einem anderen Namensraum angehören als dem, für den das Schema definiert wird. Außerdem kann definiert werden, ob eine Deklaration für die jeweiligen Elemente vorhanden sein muss (`processContents="strict"`), gemäß derer die Elemente gültig sein müssen, oder ob sie zu beachten ist, wenn vorhanden (`processContents="lax"`), oder ob sie ggf. zu ignorieren ist (`processContents="skip"`).

Für Attribute gibt es analog ein anyAttribute. Das folgende Beispiel [TBMM01] illustriert die Verwendung von Platzhaltern:

```
<element name="htmlDokument">
  <complexType>
    <sequence>
      <any namespace="http://www.w3.org/1999/xhtml"
        minOccurs="1" maxOccurs="unbounded"
        processContents="skip"/>
    </sequence>
    <anyAttribute
         namespace="http://www.w3.org/1999/xhtml"/>
  </complexType>
</element>
```

Für das Element sind XHTML-Attribute erlaubt, als Inhalt eine Folge beliebiger XHTML-Elemente.

2.4.5 Definition einschränkender Bedingungen

XML Schema erlaubt die Definition von Schlüssel-/Eindeutigkeitsbedingungen und Referenzbeziehungen, jeweils bezogen auf ein einzelnes XML-Dokument. Im Gegensatz zu den Aspekten werden hier nicht einzelne Werte eingeschränkt, sondern Bedingungen an eine Menge von Werten spezifiziert.

Eindeutigkeit von Werten

Eindeutigkeit kann unabhängig vom Typ definiert werden und wird immer anhand typbezogener Gleichheit entschieden. Zwei Werte können nicht gleich sein, wenn sie nicht von demselben Basistyp sind. Ansonsten werden sie typentsprechend verglichen. Zum Beispiel sind »3« und »3.0« gleich, wenn sie vom Typ *number* sind, aber ungleich, wenn sie vom Typ *string* sind.

Eindeutigkeit kann für einzelne Element- oder Attributinhalte, aber auch für Kombinationen darauf gefordert werden, und zwar relativ zu dem Element, in dem die entsprechende Eindeutigkeitsbeziehung definiert wird. Die zu berücksichtigenden Teile des Schemas werden über *selector*- und *field*-Elemente spezifiziert. Im *selector*-Element wird ein relativer XPath-Ausdruck[2] angegeben, der eine Knotenmenge von Knoten unterhalb des deklarierten Elements identifiziert. Jedes *field*-Element identifiziert relativ dazu genau einen Knoten. Die Kombination aller so referenzierten Werte muss für die über *selector* ausgewählte Knotenmenge innerhalb einer Instanz des deklarierten Elements eindeutig sein. Bei der Eindeutigkeit wird unterschieden zwischen *unique* (wenn der Wert vorhanden ist, muss er eindeutig sein) und *key* (wie unique, nur muss der Wert vorhanden sein).

Folgendes Beispiel definiert, dass die Namen von Kontakten innerhalb eines Adressbuchs eindeutig sein müssen:

```
<xs:element name="Adressbuch">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Kontakt" minoccurs="1" maxoccurs="unbounded">
    </xs:sequence>
  </xs:complexType>
  <xs:key name="Kontaktname">
    <xs:selector xpath="Kontakt"/>
    <xs:field xpath="@Name"/>
  </xs:key>
</xs:element
```

Referenzbeziehungen

Es gibt weiterhin die Möglichkeit, Referenzbeziehungen zu definieren (in Analogie zu den Primärschlüssel-/Fremdschlüsselbeziehungen des relationalen Modells, allerdings beschränkt auf eine Instanz). Mit *keyref* kann definiert werden, dass

2. Dabei ist nur eine Untermenge der vollen XPath-Syntax (siehe Abschnitt 3.2) erlaubt.

eine Wertekombination Fremdschlüssel ist. Der Primärschlüssel kann entweder als *key* oder als *unique* definiert sein.

Folgendes Beispiel definiert, dass alle Namen von Kontakten, die in einer Verabredung auftreten, auch im Adressbuch vorkommen müssen:

```
<xs:element name="Organizer">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Adressbuch"/>
      <xs:element name="Verabredung">
         <xs:complexType>
           <xs:sequence>
             <xs:element name="Datum" type="xs:dateTime"/>
             <xs:element name="Ort" type="xs:string"/>
             <xs:element name="Partner" type="xs:string"/>
           </xs:sequence>
         </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:keyref name="Kontaktkonsistenz" refer="Kontaktname">
    <xs:selector xpath="Verabredung"/>
    <xs:field xpath="Partner"/>
  </xs:keyref>
</xs:element>
```

Die Tatsache, dass XML-Schema durch *list* mehrere Werte eines Wertebereiches kombinieren kann, wird von *keyref* nicht reflektiert: Es ist nicht möglich, mit einem Wert in *keyref* ein einzelnes Element einer Liste, die im entsprechenden *key* steht, zu referenzieren. Umgekehrt ist es auch nicht möglich zu formulieren, dass die einzelnen Werte einer Liste verschiedene andere Werte referenzieren. Wenn also im Beispiel im Unterelement *Partner* des Elementes *Verabredung* eine Liste von Namen enthalten sein kann, kann man die Bedingung, dass jeder von diesen einem existierenden Kontakt entspricht, nicht modellieren. Somit ist eine Modellierung einer multiplen Verweisbeziehung, wie sie durch IDREFS gegeben ist, mit diesem Konzept nicht möglich.

Im Gegensatz zu Datenbankschemata gibt es in XML Schema keine Möglichkeit, instanzübergreifende Integritätsbedingungen zu definieren, also etwa Schlüsselbedingungen in einer Menge von XML-Dokumenten oder Referenzbeziehungen zwischen verschiedenen XML-Dokumenten.

2.4.6 Namensräume

Die in einem Schemadokument deklarierten Elemente und Attribute können zu höchstens einem Namensraum gehören, der im umgebenden *schema* Element im Attribut *targetNamespace* angegeben ist. Fehlt dieses Attribut, gehören die Dekla-

rationen zu keinem Namensraum. Als Konsequenz kann ein Schemadokument nicht Elemente oder Attribute verschiedener Namensräume deklarieren – dazu sind die Mechanismen des Schemaimports (s.u.) nötig.

Globale Elemente, Attribute und Definitionen gehören immer dem Namensraum des Schemas an. Lokal deklarierte Elemente und Attribute gehören ihm nur an, wenn das Attribut *elementFormDefault* bzw. *attributeFormDefault* von *schema* den Wert *qualified* hat, oder das Attribut *form* in der Element-/Attributdeklaration den Wert *qualified* hat. Sonst gehören sie keinem Namensraum an.

Das Vokabular, das von XML Schema definiert wird, wird durch den Namensraum `http://www.w3.org/2001/XMLSchema` identifiziert. In den obigen Beispielen wurde für diesen Namensraum immer das Präfix *xs:* benutzt, so als stünde in einem umgebenden Element das Attribut

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

XML Schema sieht jedoch auch vor, dass schemabezogene Attribute in jeglichen XML-Dokumenten stehen können. Diese gehören zum Namensraum `http://www.w3.org/2001/XMLSchema-instance`, für den hier das Präfix *xsi:* benutzt wird.

Die Namen, die mittels XML Schema definiert werden, gehören jeweils einem Gültigkeitsbereich an, innerhalb dessen sie eindeutig sein müssen. Jede Art von Definition und Deklaration hat ihren eigenen Gültigkeitsbereich (*symbol space*), lediglich einfache und komplexe Typen teilen sich einen Gültigkeitsbereich. Somit ist es erlaubt, ein Element genauso zu benennen wie einen einfachen Typ, nicht jedoch, einen komplexen Typ so zu nennen wie einen einfachen Typ.

2.4.7 Ableitung, Ersetzung, Import

Wie wir gesehen haben, können Typen durch Erweiterung oder Einschränkung anderer definiert werden. Da man nicht immer möchte, dass aus einem Typ ein anderer auf diese Weise abgeleitet werden kann, kann man dieses durch das Attribut *final* beschränken: Man kann jegliche Ableitung verbieten, oder gezielt Einschränkung oder Erweiterung.

Auch für Aspekte kann man verbieten, dass sie sich bei Ableitung verändern, nämlich durch Angabe von `fixed="true"`.

Ersetzungsgruppen

XML Schema führt einen neuen Mechanismus ein, der es erlaubt, statt eines in einer Typdefinition angegebenen Elements ein anderes einzusetzen. Dieser Mechanismus heißt Ersetzungsgruppe (substitution group). Trotz der Namensähnlichkeit mit Attributgruppen und Modellgruppen handelt es sich hier um ein damit nicht verwandtes Konzept. Es werden gewissermaßen Äquivalenzklassen von Elementen definiert.

Bei einer Elementdeklaration kann man angeben, dass das jeweilige globale Element als Ersatz für ein anderes globales (den so genannten Ersetzungsgruppenkopf, substitution group head) gelten kann. Alle Elemente, die dasselbe Element ersetzen, bilden eine Ersetzungsgruppe (dies setzt sich transitiv fort, falls der Ersetzungsgruppenkopf selbst wieder ein anderes Element ersetzen kann). Die Namen der Elemente und ihre Namensräume spielen dabei keine Rolle. Die Typen der Elemente müssen jedoch entweder identisch sein oder durch Einschränkung oder Erweiterung auseinander hervorgegangen sein. Ein Element kann durch Angabe des *block*-Attributes verhindern, dass es zum Ersetzungsgruppenkopf wird, oder kann erzwingen, dass im Falle einer Ersetzungsgruppe keine Erweiterung oder keine Einschränkung vorgenommen wird. Andererseits kann bei einer Elementdeklaration durch Angabe von `abstract="true"` erzwungen werden, dass das Element durch ein Mitglied der Ersetzungsgruppe ersetzt wird. Ein einfaches Beispiel:

```
<xs:element name="Kontakt">
  <complexType>
    <xs:sequence>
      <xs:element ref="Name"/>
      <xs:element ref="Telefonnummer"/>
    </xs:sequence>
  </complexType>
</xs:element>
<xs:element name="Telefonnummer"
            type="Telefonnummer"/><xs:element name="Name"
            type="xs:string"/>
<xs:element name="phone"
            type="Telefonnummer"
            substitutionGroup="Telefonnummer"/>
```

Hier kann phone für Telefonnummer eingesetzt werden. Damit validiert das XML-Fragment

```
<Kontakt>
  <Name>Hans Müller</Name>
  <phone>0123/123123</phone>
</Kontakt>
```

gegen obige Schemadefinition. Eine Motivation für dieses Konzept könnte sein, Interoperabilität zwischen Schemata unterschiedlicher Herkunft zu schaffen, die denselben Inhalt beschreiben.

Schema-Import

XML-Schemata können aus mehreren Schemadokumenten zusammengesetzt werden. Über ein *include*-Element können Schemadokumente eingefügt werden, die Deklarationen zu demselben Namensraum oder ohne Namensraumbindung enthalten.

Als Erweiterung gibt es hier die Möglichkeit, Typ- oder Gruppendefinitionen beim Übernehmen umzudefinieren (*redefine*), allerdings in verträglicher Weise. Das heißt z. B., dass ein Typ nur mit sich selbst als Basistyp redefiniert werden darf.

Schemadokumente, die einem anderen Namensraum angehören, können über import hinzugenommen werden. Dann ist es möglich, aus Deklarationen und Definitionen auch auf Dinge Bezug zu nehmen, die in diesem anderen Namensraum definiert sind. Es sei daran erinnert, dass auch *any* die Möglichkeit bietet, Teile aus anderen Namensräumen zuzulassen.

2.4.8 Zusammenhang zu DTDs

Ein Aspekt von DTD fehlt in XML Schema ganz: die Entities. XML Schema sieht zwar den Attributtyp ENTITY vor, unterstützt selbst aber keine Mechanismen zur Entity-Definition. Es scheint, als habe sich das W3C von dem Entity-Konzept verabschiedet, denn auch in anderen neueren W3C-Standards spielen Entities keine Rolle mehr. Wenn Entities zusammen mit XML Schema verwendet werden sollen, muss zusätzlich zum XML-Schema eine DTD vorhanden sein. Wie in [Wa01] erläutert, erfolgt dabei zunächst die Validierung gegen die DTD, anschließend wird gegen das XML Schema validiert. Das bedeutet, dass eine Instanz nicht validiert, wenn die Validierung gegen die DTD fehlschlägt, aber die Instanz alleine gegen das XML-Schema valide wäre.

2.4.9 Instanzbezogene Konzepte

xsi:type

Normalerweise wird der Typ eines Elementes aus der Deklaration im Schema erschlossen. Es gibt jedoch auch die Möglichkeit, in einer Instanz die Betrachtung eines Elementes als zu einem bestimmten Typ zugehörig zu erzwingen, indem dem Element im Dokument das Attribut xsi:type mit dem entsprechenden Wert hinzugefügt wird.

xsi:nil

Aus Datenbanksystemen kennt man das Konzept der Nullwerte, das sich so in XML Schema nicht wiederfindet. Es gibt jedoch ein Konzept, das diesem ähnelt: Wenn einer Elementdeklaration `nillable="true"` hinzugefügt wurde, dann wird ein Element, das keinen Inhalt hat, als gültig akzeptiert, wenn es als Attribut `xsi:nil="true"` hat, auch wenn der Typ des Elements eigentlich keinen leeren Wert erlaubt. Einige Unterschiede zu dem NULL-Konzept aus Datenbanken seien explizit erwähnt: Hier findet eine klare Unterscheidung zwischen einem nicht vorhandenen Element, einem Element mit (ggf. leerem) Inhalt aus dem Wertebereich und einem *nil*-Element statt. In relationalen Datenbanksystemen kann auch das Nichtvorhandensein eines Wertes nur durch NULL gekennzeichnet werden. Mit

dem Konzept in XML Schema ist keine Verarbeitungssemantik (wie etwa dreiwertige Logik) verbunden.

xsi:schemaLocation und xsi:noNamespaceSchemaLocation

XML Schema definiert nicht, wie das Schema zu einem XML-Dokument gefunden wird. Eine Möglichkeit, aus dem Dokument das entsprechende Schema zu identifizieren, stellt das Attribut xsi:schemaLocation dar. Ein Schema, das keinen Namensraum definiert, kann über *xsi:noNamespaceSchemaLocation* referenziert werden.

2.5 Vergleich relationaler Modellierungskonzepte mit XML Schema

Im Folgenden wollen wir die wichtigsten relationalen Modellierungskonzepte der SQL-Realisierung des relationalen Modells den entsprechenden XML-Schema-Konzepten gegenüber stellen. Wir betrachten dabei nur die in RDBMS vorhandenen Konzepte. Ein ausführlicherer Vergleich ist in [KKR01] enthalten.

2.5.1 Strukturierte Datentypen

Für die wesentlichen relationalen Typkonstruktoren findet man Entsprechungen in XML Schema. Tabellen entsprechen multiplen XML-Elementen. Die Spalten sind in RDBMS mit den Attributen in XML Schema weitgehend identisch. In beiden Fällen sind dafür nur einfache Datentypen zugelassen. Elemente sind jedoch in XML ein wesentlich mächtigeres Konzept im Vergleich zu den Tabellen. Tabellen enthalten nur Attribute. Elemente gehen in folgenden Aspekten darüber hinaus:

- Hierarchische Schachtelung von Elementen.
- Elemente können direkt Werte von einfachen Datentypen enthalten.
- Elemente können sowohl Werte als auch andere Elemente enthalten (gemischter Inhalt).

Die in SQL vorhandenen einfachen Datentypen wie *VARCHAR*, *INTEGER* usw. lassen sich vollständig auf entsprechende XML-Schema-Datentypen abbilden. Darüber hinaus enthält XML Schema noch eine größere Anzahl von zusätzlichen Datentypen wie *token* und *anyURI*.

2.5.2 Null- und Defaultwerte

Die Defaultwerte von Spalten entsprechen den Defaultwerten für Attribute in XML. In XML Schema sind zusätzlich Defaultwerte für Elemente mit einfachen Datentypen möglich. Weiterhin erlaubt XML Schema die Definition von Konstanten für Attribute und Elemente mit einfachen Datentypen.

SQL-Nullwerte lassen sich auf mehrere Arten in XML Schema definieren. Eine Möglichkeit ist die Definition von optionalen Attributen (`use="optional"`) oder Elementen (`minOccurs="0"`) in XML. Die weitere Möglichkeit nur für Elemente ist der spezielle Instanzwert xsi:nil="true". Im entsprechenden XML-Schema kann dieser durch `nillable="true"` erlaubt werden.

2.5.3 Eindeutigkeit von Werten

Der Primärschlüssel in RDBMS zur Identifikation von Zeilen ist ebenfalls als Konzept in XML Schema vorhanden. Die Unterschiede bestehen darin, dass ein Primärschlüssel in RDBMS immer eine Zeile in einer Tabelle identifiziert. Dagegen kann beim Schlüssel in XML Schema der Bereich, in dem der Schlüssel gültig ist, durch einen XPath-Ausdruck (siehe Abschnitt 3.2) festgelegt werden.

2.5.4 Referenzbeziehungen

Die Fremdschlüssel von RDBMS zur Modellierung von Beziehungen sind ebenfalls als analoges Konzept in XML Schema vorhanden. Falls man Relationen als XML-Elemente und Spalten als XML-Attribute in einem einzigen XML-Dokument realisiert, lässt sich jeder Primärschlüssel auf einen XML-Schema-*key* und jeder Fremdschlüssel auf einen XML-Schema-*keyref* abbilden.

2.5.5 Ordnung

Hier besteht ein fundamentaler Unterschied zwischen RDBMS und XML. Generell sind alle XML-Elemente in einem Dokument geordnet, während die Zeilen einer Tabelle ungeordnet sind. Die zulässige Anordnung der XML-Elemente wird durch das Inhaltsmodell des übergeordneten Elements festgelegt. Erstaunlicherweise fällt der Vergleich bei Attributen und Spalten genau umgekehrt aus. Die Attribute eines XML-Elements sind nicht geordnet. Andererseits sind die Spalten einer Tabelle geordnet. Die Ordnung von Attributen und Spalten spielt jedoch eine wesentlich geringere Rolle bei den jeweils üblichen Operationen.

2.5.6 Zusammenfassung

Die folgende Tabelle stellt die diskutierten Konzepte in RDBMS noch einmal den entsprechenden XML-Schema-Konzepten gegenüber. Insgesamt kann man feststellen, dass sich das relationale Datenmodell zwar nicht exakt, aber doch in vielen Aspekten durch XML Schema abdecken lässt. Das eröffnet eine Reihe von Abbildungsmöglichkeiten des relationalen Modells auf XML Schema. In unserer groben Gegenüberstellung haben wir allerdings auch schon auf einige der vielen Detail-Unterschiede hingewiesen, so dass konkrete Lösungen relativ kompliziert werden.

Konzept	RDBMS	XML Schema
Typkonstruktoren	Tabelle als set of tupel	Multiple Elemente mit komplexem Inhaltsmodell
Attribut	Spalte	Attribut
Einfache Datentypen	10	43
Nullwerte	null	Nillable Optionale Attribute oder Elemente
Identifikation	Primärschlüssel	key
Referenzbeziehungen	Fremdschlüssel	keyref
Ordnung	Ungeordnete Zeilen einer Tabelle (geordnete Spalten)	Geordnete Sequenzen von Elementen (Ungeordnete Attribute)

Tab. 2-1: *Vergleich der Konzepte von RDBMS und XML Schema*

2.6 Modellierungsstile

Aufgrund der vielen in XML Schema enthaltenen Modellierungskonzepte stellt sich besonders die Frage nach geeigneten Modellierungsstilen. Es ist nicht weiter verwunderlich, dass mit XML Schema unterschiedlichste Modellierungsstile verwendet werden können. Wir verstehen darunter nur den Einsatz einer bestimmten Teilmenge von XML-Schema-Konzepten gegenüber einer anderen Teilmenge von XML-Schema-Konzepten, um das gleiche Problem zu lösen.

Es geht hier nicht um Designmethoden, die eine bestimmte Qualität der Schemata, wie Redundanzfreiheit oder Normalisierung, garantieren. Solche Designmethoden sind im Gegensatz zum Relationen-Modell für XML Schema noch praktisch unbekannt. Indirekte Lösungen des Problems sind erkennbar, indem erste Abbildungen von UML auf XML Schema existieren. Für UML stehen eine Vielzahl von Designmethoden zur Verfügung. Der Nachteil einer solchen indirekten Methode ergibt sich aus dem Abbildungsverlust, der auftritt, wenn das objektorientierte, graphenartige UML-Modell in ein baumartiges XML-Schema-Modell umgesetzt wird.

2.6.1 Verwendung und Sichtbarkeit von vollständigen Elementdeklarationen

Das betrifft zunächst einmal die Frage, wie die Definition von vollständigen Elementen oder Attributen innerhalb und außerhalb eines Schemadokuments wiederverwendet werden kann. Wichtige Unterscheidungsmerkmale ergeben sich aus der Art der Verwendung von folgenden Funktionalitäten:

- Lokale versus globale Elemente.
- Verwendung von benamten einfachen und komplexen Typen.

Globale Elementmodellierung (»Salami«-Design)

Das globale Elementdesign (in Diskussionsforen auch unter dem Namen »Salami«-Design bekannt) verwendet nur global deklarierte Elemente und Attribute. Die Inhaltsmodelle der einzelnen Elemente werden dann nur mit Referenzen auf die globalen Elemente und Attribute definiert.

```
<xs:element name="TelefonKontakt">
  <complexType>
    <xs:sequence>
      <xs:element ref="Vorname" maxOccurs="unbounded"/>
      <xs:element ref="Nachname"/>
      <xs:element ref="Telefonnummer"/>
      <xs:element ref="Faxnummer" minOccurs="0"/>
    </xs:sequence>
  </complexType>
</xs:element>
<xs:element name="Vorname"
            type="xs:string"/>
<xs:element name="Nachname"
            type="xs:string"/>
<xs:element name="Telefonnummer"<
<xs:simpleType>
   <xs:restriction base="xs:string">
      <xs:pattern value="0[0-9]+/[0-9]+"/>
   </xs:restriction/>
</xs:simpleType>
</xs:element>
<xs:element name="Faxnummer"<
<xs:simpleType>
   <xs:restriction base="xs:string">
      <xs:pattern value="0[0-9]+/[0-9]+"/>
   </xs:restriction/>
</xs:simpleType>
</xs:element>
```

Ein Nachteil dieses Modellierungsstils ist, dass sämtliche Elemente global sind und damit Bestandteil des gleichen Namensraums. So können keine Element- oder Attributnamen lokal gewählt werden. Änderungen von solchen globalen Elementen wirken sich sofort auf das gesamte Schema aus. Ferner sind alle Elemente außerhalb des Schemadokuments (zum Beispiel beim Import in ein anderes Schema) sichtbar.

Dieser Modellierungsstil entspricht genau den Modellierungsmöglichkeiten mit DTDs. Dort ist er der einzig mögliche Modellierungsstil, da die DTD nur globale Elemente kennt. Hier ergeben sich keinerlei Einschränkungen bei der Verwendung von Elementen für die Definition von rekursiven Elementen, die nur auf globalen Elementen möglich sind.

Daher ist diese Modellierung eher für dokumentenorientierte Anwendungen geeignet.

Lokale Elementmodellierung (»Babuschka«-Design)

Der gegenteilige Modellierungsstil zur globalen Elementmodellierung verwendet nur lokale Elemente. Nur das Wurzelelement für eine XML-Instanz ist global. Sämtliche anderen Elemente und Attribute werden direkt bei der Verwendung in den Inhaltsmodellen der Elemente lokal komplett deklariert.

```
<xs:element name="TelefonKontakt">
  <complexType>
    <xs:sequence>
      <xs:element name="Vorname"
 maxOccurs="unbounded">
             type="xs:string"/>
<xs:element name="Nachname"
           type="xs:string"/>
<xs:element name="Telefonnummer"<
<xs:simpleType>
   <xs:restriction base="xs:string">
      <xs:pattern value="0[0-9]+/[0-9]+"/>
   </xs:restriction/>
</xs:simpleType>
</xs:element>
<xs:element name="Faxnummer" minOccurs="0">
<xs:simpleType>
   <xs:restriction base="xs:string">
      <xs:pattern value="0[0-9]+/[0-9]+"/>
   </xs:restriction/>
</xs:simpleType>
</xs:element>
    </xs:sequence>
  </complexType>
</xs:element>
```

Dieser Modellierungsstil hat den Vorteil, dass sämtliche Elemente lokal sind. So können Element- oder Attributnamen lokal gewählt und unabhängig geändert werden. Bis auf das Wurzelelement sind keine Elemente nach außen sichtbar und können auch nicht wiederverwendet werden. Das bedeutet, dass genau gleichartig aufgebaute Elemente mehrfach vollständig deklariert werden müssen. Da Rekursion nur mit globalen Elementen modelliert werden kann, müssen neben den Wurzelelementen auch noch alle rekursiven Elemente global sein. Sie sind damit auch nach außen sichtbar.

Typ-Modellierung (»Jalousie«- Design)

Die Typ-Modellierung kann als Variante der lokalen Elementmodellierung gesehen werden. Hier werden ebenfalls nur lokale Elemente deklariert. Allerdings werden globale benamte Typen verwendet, wo Inhaltsmodelle Gemeinsamkeiten haben.

```
<xs:element name="TelefonKontakt">
  <complexType>
    <xs:sequence>
      <xs:element name="Vorname"
 maxOccurs="unbounded">
              type="xs:string"/>
<xs:element name="Nachname"
            type="xs:string"/>
<xs:element name="Telefonnummer"
 type="Telefonnummer"/>
<xs:element name="Faxnummer"
              type="Telefonnummer">
 minOccurs="0"/>
    </xs:sequence>
  </complexType>
</xs:element>
<xs:simpleType type="Telefonnummer">
   <xs:restriction base="xs:string">
      <xs:pattern value="0[0-9]+/[0-9]+"/>
   </xs:restriction/>
</xs:simpleType>
```

Hier sind globale Typen nach außen sichtbar und können wiederverwendet werden. Die Elemente selbst sind bis auf das Wurzelelement alle lokal und damit nicht sichtbar. Somit kann von Fall zu Fall entschieden werden, was nach außen sichtbar und wiederverwendbar sein soll. Die Typ-Modellierung entspricht damit eher der üblichen Modellierung von Datenstrukturen in Programmiersprachen.

Bei der Typ-Modellierung sind allerdings globale Elemente nicht nur für Wurzelelemente, sondern auch für rekursive Elemente erforderlich. Die Rekursion ist in XML Schema nur auf globalen Elementen und nicht auf Typen oder lokalen Elementen möglich.

Generell ist sie damit eher für datenorientierte Anwendungen geeignet.

2.6.2 Teile von Elementdeklarationen

Bisher haben wir betrachtet, wie vollständige Elementdeklarationen definiert werden und wiederverwendet werden können. Durch die umfangreichen Modellierungsmöglichkeiten der Inhaltsmodelle ist es jedoch notwendig, auch Teile von Elementdeklarationen wiederzuverwenden. Dafür gibt es in XML Schema zwei prinzipielle Möglichkeiten:

- Gruppen-Modellierung: Verwendung von Element- oder Attributgruppen.
- Programmiersprachen-orientierte Modellierung: Vererbungsmechanismen zwischen Typdefinitionen.

Gruppen-Modellierung

Gruppen erlauben, dass man Teilen von Inhaltsmodellen einen Namen gibt und diese benamten Gruppen zum Aufbau von vollständigen Inhaltsmodellen verwenden kann. Sie sind damit eher mit einem Makromechanismus zu vergleichen.

```
<xs:element name="Vorname"
            type="xs:string"/>
<xs:element name="Nachname"
            type="xs:string"/>
<xs:element name="Telefonnummer"
 type="Telefonnummer"/>
<xs:element name="Faxnummer"
              type="Telefonnummer">

<xs:group name="Kontakt">
   <xs:sequence>
      <xs:element ref="Vorname" maxOccurs="unbounded"/>
<xs:element ref="Nachname"/>
    </xs:sequence>
</xs:group>

<xs:element name="Kontakt">
  <complexType>
     <xs:sequence>
       <xs:group ref="Kontakt"
</xs:sequence>
   </complexType>
</xs:element>

<xs:element name="TelefonKontakt">
  <complexType>
    <xs:sequence>
      <xs:group ref="Kontakt"/>
<xs:element ref="Telefonnummer"/>
<xs:element ref="Faxnummer" minOccurs="0"/>
    </xs:sequence>
  </complexType>
</xs:element>
```

Programmiersprachen-orientierte Modellierung

Einen ähnlichen Nutzen erreicht man mit der Ausnutzung der in XML Schema enthaltenen Vererbungsmechanismen. Zusätzlich ist aber damit eine Typprüfung für die Instanzen verbunden. Dieser Modellierungsstil entspricht so weitgehend

den Modellierungsmöglichkeiten von objektorientierten Programmiersprachen. Im folgenden Beispiel wird der dort übliche Vererbungsmechanismus der Erweiterung des Zustandes beziehungsweise der Schnittstelle benutzt. Der Subtyp »TelefonKontakt« wird von dem Typ »Kontakt« abgeleitet, indem er die zusätzlichen Elemente »Telefonnummer« und »Faxnummer« bekommt.

```
<xs:complexType name="Kontakt">
   <xs:sequence>
      <xs:element name="Vorname"
 maxOccurs="unbounded">
              type="xs:string"/>
<xs:element name="Nachname"
           type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<xs:element name="Kontakt" type="Kontakt"/>

<xs:complexType name="TelefonKontakt">
    <xs:extension base="Kontakt">
      <xs:sequence>
  <xs:element name="Telefonnummer"
   type="Telefonnummer"/>
  <xs:element name="Faxnummer"
               type="Telefonnummer">
   minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
</xs:complexType>

<xs:element name="TelefonKontakt" type="TelefonKontakt"/>
```

Allerdings gehen die strukturellen Vererbungsmechanismen von XML Schema stark über die strukturellen Vererbungsmechanismen[3] in objektorientierten Programmiersprachen hinaus. Neben der Vererbung durch Erweiterung gibt es noch eine Vererbung durch Einschränkung, indem für den Subtyp ein Inhaltsmodell angegeben wird, das den Wertebereich einschränkt. Dazu muss das gesamte Inhaltsmodell für den Subtyp angegeben werden.

3. Objektorientierte Programmiersprachen haben zusätzlich starke verhaltensmäßige Vererbungsmechanismen. Da XML Schema überhaupt kein Verhalten abdeckt, sind diese in XML Schema natürlich nicht vorhanden.

2.7 Physisches Schema

Datenbanksysteme haben neben dem logischen Schema, wie es mit XML Schema gegeben ist, im Allgemeinen auch den Bedarf für ein physisches Schema, in dem z. B. Indexe definiert sind oder Speicherungsvorschriften angegeben sind. Für XML benötigen relationale Datenbanken beispielsweise oft Abbildungsvorschriften auf Tabellen. Diese physische Information ist typischerweise auf Schemaelemente bezogen – es wäre also sinnvoll, diese auch explizit an die Schemaelemente anbinden zu können. Hierfür gibt es mehrere Möglichkeiten

Entkoppelte Information

Die physische Information wird in einem eigenen Dokument abgelegt, das lediglich die Namen aus der XML-Schemadefinition übernimmt. Ein Beispiel hierfür sind die DAD-Dateien (Data access definition) von DB2, in denen die Abbildung von XML-Dokumenten auf Tabellen beschrieben ist. Dieser Ansatz bietet keine Mechanismen, die bei der Änderung des XML-Schemas entstehende Inkonsistenzen verhindern oder aufzeigen.

Proprietäre Schemasprache

Wenn statt einer normierten Schemasprache eine proprietäre Sprache verwendet wird, kann diese auch Konstrukte enthalten, die physische Informationen für die jeweilige Anwendung beschreiben. Ein Beispiel hierfür ist die Definitionssprache von XML Views in Microsoft SQL Server.

Schema Adjunct

Einen allgemeinen Rahmen für eine Schemaanreicherung um proprietäre Informationen bietet das *schema adjunct framework* [SA02]. In einem separaten Dokument werden diese Informationen abgelegt. Da es sich um einen allgemeinen, anwendungsunabhängigen Rahmen handelt, kann ein entsprechendes Dokument von einem allgemeinen Werkzeug zur Verarbeitung von XML-Schemata konsistent gehalten werden.

Schemasprachenerweiterung

XML Schema bietet einen Mechanismus, jegliche proprietäre Information direkt in das Schema einzubetten. Zu jeder Definition und Deklaration kann ein *annotation*-Element hinzufügt werden. Dieses hat zwei mögliche Unterelemente, eines für Benutzerinformation (*documentation*) und eines für maschinell auswertbare Information (*appinfo*). Hier angebrachte Information kann von allen Applikationen ignoriert werden, die diese nicht verstehen. Beispiele für die Verwendung von appinfo sind SQL/XML [SQLX] und die aktuelle Schemasprache des Datenbanksystems Tamino. Die entsprechenden Elemente unterhalb von appinfo gehören natürlich nicht zum XML-Schema-Namensraum (im folgenden Beispiel sei für diesen das Präfix tsd definiert):

```
<xs:element name="Name" type="string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:native>
            <tsd:index>
              <tsd:text/>
            </tsd:index>
          </tsd:native>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Trotz der Anreicherung um proprietäre Information bleibt dieses XML-Schema für alle XML-Schemaprozessoren verständlich. Es ist darüber hinaus möglich, dass verschiedene proprietäre Informationen gleichzeitig hinzugefügt werden.

2.8 Zusammenfassung

XML Schema bietet im Gegensatz zu DTD mächtigere Modellierungskonzepte. Dazu gehören ein umfangreiches Typsystem, die Möglichkeit, lokale Definitionen vorzunehmen, ein Vererbungsmechanismus und Beschränkungen. XML Schema ermöglicht damit die Verwendung verschiedener Modellierungsstile. Diese Vielfalt wird mit einer sehr hohen Komplexität des W3C-XML-Schema-Standards erkauft.

Als Schemasprache zur Beschreibung von XML in Datenbanken stellt XML Schema jedoch trotz seiner Komplexität einen großen Fortschritt dar. Dies ergibt sich daraus, dass es alle benötigten Funktionalitäten enthält und dass es ein W3C-Standard ist, der immer mehr in darauf aufbauenden W3C-Standards Verwendung findet. Aus diesen Gründen wird XML Schema heute in kommerziellen Produkten in zunehmender Weise unterstützt.

Literatur

[BiMa01] Biron, P.V., Malhotra, A. (Hrsg.): *XML Schema Part 2: Datatypes*, W3C Recommendation 02 May 2001. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502.

[BPSM00] Bray, T., Paoli J., Sperberg-McQueen, C. M., Maler, E. (Hrsg.): *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation 6 October 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[BrHL99] Bray, T., Hollander, D., Layman, A. (Hrsg): *Namespaces in XML*, W3C Recommendation, January 14, 1999. http://www.w3.org/TR/1999/REC-xml-names- 19990114.

[CM01] Clark, J., Makoto, M.: *RELAX NG Specification*, OASIS Committee Specification 3 December 2001.
http://www.oasis-open.org/committees/relax-ng/spec-20011203.html.

[Fall01] Fallside, D.C. (Hrsg.): *XML Schema Part 0: Primer*, W3C Recommendation, 2 May 2001. http://www.w3.org/TR/2001/REC-xmlschema-0-20010502.

[KKR01] Kappel, G., Kapsammer, E., Retschitzegger, W.: *XML and Relational Database Systems - A Comparison of Concepts*, International Conference on Internet Computing (1) 2001.

[LC00] Lee, D., Chu W.: *Comparative Analysis of Six XML Schema Languages*. In: ACM SIGMOD Record, Vol.29, Nr. 3 September 2000.

[SA02] *The Schema Adjunct Framework*.
http://www.tibco.com/products/extensibility/resources/saf.htm.

[SQLX] Jim Melton (Editor): (ISO-ANSI Working Draft) *XML-Related Specifications* (SQL/XML), s.a. http://www.sqlx.org.

[TBMM01] Thompson, H.S., Beech, D., Maloney , M., Mendelsohn, N. (Hrsg): *XML Schema Part 1: Structures*, W3C Recommendation 2 May 2001.
http://www. w3.org/TR/2001/REC-xmlschema-1-20010502.

[Wa01] Walmsley, P: *Definitive XML Schema*, Prentice Hall, 2001.

3 Anfragen, Ändern und Transformieren von XML

Georg Lausen, Wolfgang May

Kurzfassung

Dieses Kapitel hat Anfragesprachen, Datenmanipulation und -transformation von XML-Daten zum Inhalt. Dabei wird sowohl die Entwicklung der Konzepte beschrieben als auch eine Einführung in die gegenwärtig populärsten Sprachen, XPath und XQuery, gegeben. XML-Anfragesprachen gehen zurück auf die frühen XSL Patterns und XQL, aus denen sich XPath als Adressierungsformalismus entwickelte, der die Grundlage für mächtigere Sprachen zu XML bildet. In diesem Kapitel werden die Anfragesprachen XML-QL sowie das auf XPath basierende Quilt, aus dem dann XQuery hervorging, beschrieben. Mittlerweile wurden zu diesen bis dahin reinen Anfragesprachen auch Konzepte zur Datenmanipulation vorgeschlagen, die als Spracherweiterung zu XQuery auch bereits implementiert sind. Weiterhin wird in diesem Kapitel die Transformation von XML-Daten beschrieben, die letztlich eine Grundlage für die Präsentation in HTML ist.

3.1 Überblick

XML ist ein Datenmodell, das Daten in Form von Bäumen strukturiert. Anfragesprachen für XML stehen somit in starkem Kontrast zur relationalen Anfragesprache SQL, deren Grundlage Relationen sind, die nicht weiter strukturierte Mengen von Tupeln darstellen. Sprachen für XML sind somit von Natur aus komplexer als Sprachen für Relationen. Dennoch war es von Beginn der Entwicklungen an ein Ziel, mit SQL vergleichbare Anfrage- und Datenmanipulationssprachen auch für XML zu haben. Ausdrücke einer XML-Anfragesprache müssen somit deklarativ sein, vergleichbar zu `SELECT-FROM-WHERE`-Klauseln in SQL, die nur spezifizieren, *was* selektiert werden soll, nicht aber, *wie* dies intern geschehen soll. Weiterhin sollen XML-Anfragesprachen im Sinne des Datenmodells adäquat sein, also alle Eigenschaften des Datenmodells unterstützen. Für XML bedeutet dies, dass sowohl die in XML 1.0 definierten Datentypen als auch die in XML Schema definierten *complexTypes* unterstützt werden müssen. In der Entwicklung von Sprachen für XML hat sich schon früh gezeigt, dass der *Navigation* eine besondere Bedeutung zukommt: Im Hinblick auf die Baumstruktur eines XML-Dokumentes muss Navigation innerhalb des Baumes entlang unter-

schiedlicher *Achsen* (Vorgängerknoten, Nachbarknoten, Nachfolgerknoten etc.) sowie das Verfolgen von Referenzen innerhalb eines Dokuments und zwischen unterschiedlichen Dokumenten sprachlich unterstützt werden.

Heute basieren fast alle Sprachen in der XML-Welt auf XPath [XPath01] als Navigationsformalismus. Die ersten Vorgänger von XPath wurden im Rahmen der *W3C XSL Pattern Language* [XSL01] entwickelt. Zur selben Zeit prägten die *W3C XPointer* Working Drafts [XPtr01] den Begriff des *location path*. Der erste W3C XPath Working Draft von Juli 1999 [XPath99] vereinte die bis dahin in XPointer und XSL Pattern entwickelten Navigationskonzepte zu *XPath*, das seitdem als gemeinsame Basis für *W3C XSLT* (WD Juli 1999), *W3C XPointer* (WD Dez. 1999), *W3C XQuery* [XQuery01] (Februar 2001, siehe Abschnitt 3.5) sowie weitere Sprachkonzepte dient. In Abschnitt 3.2 gehen wir auf XPath ausführlich ein.

Die Entwicklung zu der sich derzeit in der Standardisierung befindlichen Anfragesprache *XQuery* begann 1998 mit *XQL* (*XML Query Language*) [XQL98, XQL99] und *XML-QL* [DFFL98a, DFFL99]. Im Jahr 2000 wurde *Quilt* [ChRF00] als erste Anfragesprache präsentiert, die die XPath-Syntax in höhere Programmierkonstrukte einbettet. Anfragen in Quilt bestehen aus einer Folge von `FOR-LET-WHERE-RETURN`-Klauseln (FLWR) mit eingebetteten XPath-Ausdrücken. Das Design von Quilt war von mehreren Programmiersprachen beeinflusst, u.a. von dem funktionalen Design von SQL und insbesondere OQL, außerdem von der Art, Knoten zu adressieren, von XPath und XQL, sowie von XML-QL. Ein wichtiger Unterschied gegenüber XML-QL ist, dass Quilt XPath-Ausdrücke anstatt XML-Patterns zur Erzeugung von Variablenbindungen verwendet. Durch die Integration von XPath in das höhere Konstrukt des FLWR-Ausdrucks gibt es im Allgemeinen mehrere äquivalente, syntaktisch grundverschiedene Anfragen, woraus sich eine direkte Anwendung der *XML Query Algebra* ergibt. Mit einigen kleinen Veränderungen wurde die Definition von Quilt (inzwischen XPath 2.0 benutzend) im Februar 2001 für die *W3C XQuery Language* [XQuery01] übernommen (siehe Abschnitt 3.5). Die Anpassung von Quilt/XQuery (bis dahin auf dem Typsystem von XML Schema basierend) an die XML Query Algebra und ihr Typsystem resultierte in dem *W3C XQuery 1.0 and XPath 2.0 Data Model* [XQPDM01] und der *W3C XQuery Formal Semantics* [XQFS01] in 2001. Außerdem wurde eine XML-Syntax für XQuery – wie durch die *XML Query Requirements* [XMLQR01] gefordert – als *W3C XQueryX* [XQX01] definiert.

Beispiel. Die in den folgenden Abschnitten vorgestellten Konzepte werden anhand eines durchgängigen Beispiels illustriert, das einen Ausschnitt aus der Mondial-Datenbank [Mondial] darstellt:

```
<mondial>
  <country car_code="B" capital="cty-Brussels" memberships="org-eu org-nato ..">
     <name>Belgium</name>
     <population>10170241 </population>
```

```
    <province id="pr-Brabant">
       <area>3358</area>
       <population>2253794</population>
       <city id="cty-Brussels" country="B">
           <name>Brussels</name>
           <population year="95">951580</population>
       </city>
       :
    <province/>
    :
  </country>
  :
  <organization id="org-eu" seat="cty-Brussels">
     <name>Europ. Union</name>
     <abbrev>EU</abbrev>
     <members type="member" country="GR F E A D I B L ..."/>
     <members type="applicant" country="AL CZ ..."/>
  </organization>
  <organization id="org-nato" seat="cty-Brussels" ...>
  :
  </organization>
  :
</mondial>
```

□

Das Kapitel ist wie folgt aufgebaut. Wir beginnen in Abschnitt 3.2 mit einer detaillierten Diskussion von XPath. Abschnitt 3.3 zeigt die Entwicklung zu XPath. In Abschnitt 3.4 führen wir in die theoretischen Hintergründe von XML-Anfragesprachen ein und stellen anschließend in Abschnitt 3.5 XQuery ausführlich dar. Die anschließenden Abschnitte 3.6 und 3.7 behandeln das Verändern von XML-Dokumenten und ihre Transformation. Verwandte Ansätze werden in Abschnitt 3.8 beschrieben. Eine Zusammenfassung und ein Ausblick schließen dieses Kapitel ab.

3.2 XPath

XPath [XPath99, XPath01] definiert den grundlegenden Mechanismus zur Adressierung von Knoten in XML-Instanzen und wird als solcher von den meisten XML-Anfragesprachen (und weiteren Konzepten) genutzt.

3.2.1 Grundlagen

Das zentrale Konzept in XPath sind *location paths*: Ein *location path* beschreibt auf deklarative Weise eine Menge von Knoten in einer XML-Instanz.

Die Syntax orientiert sich an der UNIX-Directory-Notation:

```
/mondial/country/city/name
```

selektiert beispielsweise alle Knoten *N*, so dass N ein *name*-Subelement eines *city*-Elements ist, das wiederum ein Subelement eines *country*-Subelements eines (des einzigen) *mondial*-Subelements des Wurzelelements ist.

Navigation: Axis Steps

XPath basiert auf Navigation in einem XML-Dokument mit Hilfe von Pfadausdrücken der Form `/step/step/.../step`. Die Eingabe zu jedem solchen *axis step* ist eine *Folge* von Knoten (die sich aus der Auswertung der vorhergehenden Schritte ergibt; für den ersten Schritt enthält diese Folge nur den Wurzelknoten des Dokuments). Der verbleibende Pfad wird dann für jeden Knoten dieser Folge ausgewertet, die Ergebnismengen aufgesammelt und zusammen als Ergebnis zurückgegeben. Bei der Auswertung jedes Schrittes enthält der *Auswertungskontext* die gesamte zur Auswertung des einzelnen Schrittes relevante Information. Dies ist zum einen die *Folge* von Knoten, die sich aus der Auswertung der vorhergehenden Schritte ergibt, innerhalb derer der *Kontextknoten* jeweils der aktuelle Knoten ist, zum anderen ist es die *Kontextposition*, die die Position des aktuellen Knotens im Kontext angibt. Das *Kontextdokument* ist das Dokument, das gerade verarbeitet wird. Mit dieser Information wird für den jeweiligen Kontextknoten die Knotenmenge berechnet, die sich als Ergebnis der Auswertung des einzelnen Schritts ergibt. Diese bildet dann den Kontext für den nächsten Schritt.

Jeder einzelne Schritt ist von der Form

```
axis::nodetest StepQualifiers
```

und beschreibt, dass der Schritt entlang der spezifizierten Achse in dem XML-Baum verläuft. Die Achse spezifiziert das Verhältnis zwischen dem aktuellen Kontextknoten und den in dem Schritt zu betrachtenden Knoten innerhalb des Baumes. Entlang der gewählten Achse spezifiziert der *nodetest* weiterhin den zu betrachtenden Knotentyp bzw. -namen. Von den so selektierten Knoten werden diejenigen ausgewählt, die die angegebenen *StepQualifiers* erfüllen (in früheren Versionen als *filter* bezeichnet; dieser ist in den meisten Fällen durch Prädikate über XPath-Ausdrücken gegeben; siehe unten).

Achsen

Für jeden Navigationsschritt spezifiziert die Achse die Navigationsrichtung in dem XML-Baum, ausgehend von dem gegenwärtigen Kontextknoten. *Vorwärtsachsen* (mit (f) bezeichnet) zählen die Knoten in Dokumentordnung auf, während *Rückwärtsachsen* (b) ihre Knoten in umgekehrter Dokumentordnung aufzählen. Für ein gegebenes Element definiert damit jede Achse eine Folge von Knoten:

- `self` (f): enthält genau das Element selber,
- `child` (f): zählt alle direkten Subelemente des Elements auf,
- `descendant` (f): zählt alle Subelemente in einem Tiefensuchedurchgang auf (jeweils die Wurzel eines Teilbaums, bevor der Teilbaum traversiert wird),

- `parent` (f): enthält genau den Elternknoten des Elements,
- `ancestor` (b): zählt alle Vorgängerknoten auf, den Elternknoten zuerst,
- `following-sibling` (f): zählt alle nachfolgenden Geschwisterknoten des Elements auf,
- `preceding-sibling` (b): zählt alle vorhergehenden Geschwisterknoten des Elements auf,
- `following` (f): zählt alle Knoten auf, die dem Element in der Dokumentordnung nachfolgen,
- `preceding` (b): zählt alle Knoten auf, die dem Element in Dokumentordnung vorangehen und keine Vorgänger von ihm sind (rückwärts),
- analog, `descendant-or-self` (f), `ancestor-or-self` (b),
- `attribute`: zählt alle Attribute des Elements auf, dabei ist die Ordnung nicht relevant,
- `namespace`: zählt alle Namensräume auf, die in dem aktuellen Knoten bekannt sind. Im Folgenden werden wir Namensräume nicht weiter betrachten, da sie für das Verständnis von XPath und Anfragen allgemein nicht zentral sind.

Die am häufigsten verwendeten Achsen können wie folgt abgekürzt werden:

- *path*`/nodetest` für *path*`/child::nodetest`,
- *path*`//nodetest` für *path*`/descendant-or-self/child::nodetest`, und
- *path*`/@nodetest` für *path*`/attribute::nodetest`.

Nodetests

Der zweite Bestandteil eines Schrittes ist der *nodetest*. Er schränkt den Knotentyp und -namen der in dem Schritt auszuwählenden Knoten ein. Jede Achse hat einen *principal node type*: Entlang der Attribut-Achse ist dieser »*attribute*«, für alle weiteren Achsen »*element*«, einschließlich der Text-Elemente. Die häufigsten *nodetests* sind wie folgt:

- ein Name, z.B. in *path*`/child::city/...`, der alle Subelemente des Kontextknotens mit dem angegebenen Elementnamen (hier: alle *city*-Elemente) auswählt,
- der Test auf Textinhalt durch `text()`, wobei alle Text-Subelemente des Kontextknotens selektiert werden,
- der Test auf Elementinhalt durch `node()`, wobei sich alle Nicht-Text-Subelemente des Kontextknotens qualifizieren, oder
- * (wildcard), der von allen Elementknoten erfüllt wird.

Anmerkung: Der Ausdruck *path*`//nodetest` ist nicht äquivalent zu *path*`/descendant::nodetest`. Für einen Knoten *n* der Ergebnismenge von *path* selektiert der Ausdruck *path*`//node()[3]` (welcher äquivalent ist zu *path*`//descendant-or-self/child::node()[3]`) alle dritten Kinder jedes Knotens *x* auf der *descendant-or-self*-Achse (der Kontext, bezüglich dem »[3]« ausgewertet wird, sind jeweils die Kinder von *x*) bezüglich *n*. *path*`/descendant::node()[3]` dagegen selektiert den dritten Knoten eines Tiefensuchedurchgangs ausgehend von *n*.

StepQualifiers

Die soweit erhaltene Folge von Knoten kann durch *StepQualifiers* weiter modifiziert werden. Sind mehr als ein *StepQualifier* gegeben, werden diese iterativ ausgewertet. Ein *StepQualifier* kann von der Form [*condition*] sein, womit die Folge auf diejenigen Knoten eingeschränkt wird, die diese Bedingung erfüllen, oder die Modifikation ist als Dereferenzierung gegeben.

Bedingungen als StepQualifiers

In diesem Fall enthält der *StepQualifier* eine Bedingung über XPath-Ausdrücken, die wie folgt gebildet werden kann:

- boolesche Ausdrücke über Prädikaten,
- atomare Prädikate, z.B. Vergleiche,
- Literale (Strings, numerische Werte),
- Ausdrücke und Funktionsaufrufe: z.B. arithmetische Ausdrücke, Aggregierungsfunktionen, Stringfunktionen, Datenkonvertierungsfunktionen oder Funktionen, die das Verhältnis zwischen dem aktuellen Kontextknoten und dem Kontext beschreiben:
  - `last()`: gibt die Mächtigkeit des aktuellen Kontexts zurück,
  - `position()`: ergibt den Index des Kontextknotens innerhalb des aktuellen Kontexts (z.B. »*5*«, wenn der Kontextknoten der fünfte Knoten der Knotenfolge ist, die sich aus der Auswertung des *nodetest* ergab). Prädikate, die `last()` oder `position()` enthalten, werden als *proximity position predicates* bezeichnet. Der *StepQualifier* `[position()=`*i*`]` kann als [*i*] abgekürzt werden.
  - `count(`*path*`)`: ergibt die Anzahl der Knoten, die – ausgehend vom aktuellen Kontextknoten – durch *path* adressiert werden; z.B. ergibt `count(city)` die Anzahl der direkten *city*-Subelemente des Kontextknotens.
- Pfadausdrücke: Pfadausdrücke werden als Prädikate ausgewertet die »true« ergeben, falls sie eine nicht leere Ergebnismenge adressieren.

Bei der Auswertung von *path*`/`*axis*`:`*nodetest*`[`*condition*`]` wird die Ergebnisfolge von *path* als *äußerer Fokus* bezeichnet. Für jedes Element *x* dieser Folge bildet die Ergebnisfolge der Auswertung von *x*`/`*axis*`:`*nodetest* den *inneren Fokus*. Dieser wiederum bildet den Auswertungskontext für die Auswertung von *condition* für jeden Knoten *y* des inneren Fokus als Kontextknoten.

Innerhalb der Prädikate können *relative* oder *absolute* Pfadausdrücke verwendet werden:

- Relative Pfadausdrücke werden bezüglich des aktuellen Kontextknotens ausgewertet, z.B. wird für
  $path/axis_1::nodetest_1[axis_2::nodetest_2...]$
  mit *y* wie oben der Ausdruck $y/axis_2::nodetest_2$ ausgewertet.

- Analog zu der UNIX-Directory-Notation beginnen absolute Pfadausdrücke mit »/«. Sie werden bezüglich des Wurzelknotens des XML-Dokumentes ausgewertet, z.B. bestimmt der folgende Ausdruck alle Städte, die denselben Namen wie ein Land haben:
  `//city[name/text() = /mondial/country/name/text()]`.

Vergleichsausdrücke

Die binären Vergleichsoperatoren, z.B., =, <, >, werden auf (Knoten)mengen erweitert: $expr_1$ *op* $expr_2$ ist erfüllt, wenn es einen Knoten x_1 in der Ergebnismenge von $expr_1$ und einen Knoten x_2 in der Ergebnismenge von $expr_2$ gibt, so dass x_1 *op* x_2 gilt. Hier können außerdem die in *W3C XQuery 1.0 and XPath 2.0 Functions and Operators* definierten Vergleichsoperatoren (u.a. für Tiefengleichheit sowie Vergleiche bezüglich der Dokumentordnung und verschiedene Konvertierungen) angewendet werden.

Die folgenden Beispiele beschreiben einige XPath-Ausdrücke anhand der Mondial-Datenbank.

- Der absolute Pfadausdruck `/mondial/country//city/name` selektiert alle direkten *name*-Subelemente von (möglicherweise indirekten) city-Subelementen direkter *country*-Subelemente von direkten *mondial*-Subelementen des Wurzelelementes.
- `/mondial/country//city/name/text()` selektiert den Textinhalt dieser Elemente.
- `/mondial/country[name = "Germany"]//city/name/text()` und `/mondial/country[name/text() = "Germany"]//city/name/text()` berechnen dasselbe, eingeschränkt auf deutsche Städte. Im ersten Fall wird implizit der Textinhalt des *name*-Elements mit dem String »*Germany*« verglichen.
- `//city[population > 5000000]/name/text()` selektiert alle Namen von Städten mit mehr als 5.000.000 Einwohnern. Wie oben wird implizit der Textinhalt des *population*-Subelements des *city*-Elements verglichen.
- Die Anfrage `//city[population[@year < 1990] > 5000000]/name/text()` bestimmt alle Städte, die die obige Bedingung bereits vor 1990 erfüllt haben. Hier werden die *population*-Subelemente simultan als Integer-Literale (im Vergleich) und als Elementknoten (deren *year*-Attribut verwendet wird) intepretiert.
- `/mondial/country/@car_code` gibt alle *car_code*-Attributknoten von *country*-Elementen zurück.
- `/mondial/country[inflation]` selektiert alle *country*-Elemente (d.h. die gesamten Teilbäume), die ein direktes *inflation*-Subelement besitzen.

Dereferenzierungen

Wenn die nach der Auswertung des *nodetest* erhaltene Knotenmenge aus `IDREF(S)` besteht (z.B. `//country/@capital`), kann als *modifizierender StepQualifier* der Dereferenzierungsoperator »`=>` *nameTest*« angewendet werden. Das Ergebnis ist

die Menge der Knoten mit den entsprechenden IDs – falls sie den angegebenen Elementnamen haben. So wertet z.B.

```
//country/@capital=>city
```

die @*capital*-Referenzattribute der *country*-Elemente aus, um die entsprechenden *city*-Elemente zu erhalten, die die Hauptstädte repräsentieren. Die Ergebnisfolge wird dann entsprechend der Dokumentordnung geordnet (d.h. im Allgemeinen nicht in der Reihenfolge zurückgegeben, wie sie im Zuge der Auswertung aufgezählt wurde).

Dereferenzierung ist nur dann möglich, wenn bekannt ist (z.B. durch eine DTD oder XML-Schema-Spezifikation), welche Attribute als `ID` und `IDREF(S)` deklariert sind.

Der folgende Ausdruck verwendet einen absoluten Pfad in einem *StepQualifier*, um durch ein Semi-Join die Stadt zu bestimmen, deren ID der Wert des *seat*-`IDREF`-Attributes der Organisation ist, deren Name »EU« ist:

```
//city[@id = /mondial/organization[name="EU"]/@seat]
```

An dieser Stelle soll zum Vergleich erwähnt werden, dass

```
/mondial/organization[name="EU"]/@seat
```

nicht dieses *city*-Element, sondern nur das als `IDREF` deklarierte *seat*-Attribut des die EU repräsentierenden Elements (d.h. den Wert »cty-brussels«) selektiert. Der Pfadausdruck

```
/mondial/organization[name="EU"]/@seat=>city
```

wiederum selektiert – dank Dereferenzierung – wieder das obige *city*-Element.

Das *memberships*-Attribut von *country* ist als `IDREFS` deklariert, ist also mehrwertig:

```
/mondial/country[@car_code="D"]/@memberships
```

selektiert das gesamte `IDREFS`-Attribut, d.h. den String »org-EU org-NATO...«. Die Dereferenzierung

```
mondial/country[@car_code="D"]/@memberships=>organization
```

bestimmt dann alle Elemente, die eine dieser IDs haben (und vom Typ *organization* sind), d.h. alle Organisationen, in denen Deutschland Mitglied ist.

Der Dereferenzierungsoperator ist bereits eine (ursprünglich von Quilt eingeführte) Erweiterung von XPath 2.0 gegenüber XPath 1.0, wo die `id(.)`-Funktion verwendet wurde, die häufig zu verwirrenden Ausdrücken führte.

Man betrachte die Anfrage »Bestimme alle Namen von Städten, die Sitz einer Organisation und gleichzeitig Hauptstadt eines ihrer Mitgliedsländer sind.«

Unter Verwendung der `id(.)`-Funktion in XPath 1.0 wird die Navigation entlang der verschiedenen Referenzattribute unübersichtlich:

```
id(//organization[id(./@seat) =
                  id(id(./members/@country)/@capital)]/@seat)/name/text()
```

Der äquivalente Ausdruck in XPath 2.0 unter Benutzung des Dereferenzierungsoperators ist

```
//organization[@seat=>city =
              members/@country=>country/@capital=>city]/@seat=>city/name/text()
```

Anmerkung zur Anwendung der StepQualifiers

Iterativ auf ein Zwischenergebnis anzuwendende *StepQualifiers*, die Bedingungen ausdrücken, können im Allgemeinen vertauscht bzw. zu einem einzigen *StepQualifier* zusammengefasst werden. Dies ist allerdings im Allgemeinen nicht möglich, wenn sie *proximity position predicates* enthalten.

Verallgemeinerte Schritte

Bis hierhin wurden *StepQualifiers* nur auf einzelne Schritte angewendet, wo die Eingabe zu dem *StepQualifier* sich aus der Auswertung von *axis*`::`*nodetest* sowie möglicherweise vorhergehenden *StepQualifiers* ergab. *StepQualifiers* können außerdem auf Ergebnisfolgen angewendet werden, die aus der Auswertung einer Folge von Einzelschritten – als *verallgemeinerte Schritte* bezeichnet – erhalten werden. Dieses sind Ausdrücke von der Form

`(`*step*`/.../`*step*`)`*StepQualifier* oder `(`*step*`|`*step*`)`*StepQualifier*

wobei Einzelschritte im Stil von (eingeschränkten) regulären Ausdrücken vorkombiniert werden. Häufig können diese durch Anwendung von Assoziativität und Distributivität äquivalent in die oben beschriebene »kanonische« Form transformiert werden. Auch dies ist jedoch im Allgemeinen nicht möglich, wenn sie *proximity position predicates* enthalten.

Der Ausdruck `(//country/city)[1]` selektiert beispielsweise die erste Stadt in dem ersten Land, während `//country/city[1]` für jedes Land die erste Stadt selektiert.

3.2.2 Weitere Funktionalität

XPath bietet weitere Operationen zur Konstruktion und Einschränkung von Folgen, Arithmetik, Vergleiche sowie logische Verknüpfungen von Prädikaten an.

Variablen

Variablen (geschrieben als `$`*var*) können in XPath an Literale, Knoten, Folgen von Knoten, und sogar Namen (um z.B. in *nodetests* verwendet zu werden) gebunden werden.

In XPath 1.0 konnten Variablen nur verwendet werden in Form von *variable references* auf »Umgebungs«-Variablen, die in umgebenden Befehlen einer Sprache, in die XPath eingebettet ist – z.B. XSLT oder XQuery (siehe Abschnitt 3.5) – gebunden wurden. Bei der Auswertung von XPath-Ausdrücken mit freien Variablen sind die Variablenbindungen Teil des Auswertungskontextes.

Beispiel – XPath-Ausdruck mit Variablen. Seien die Umgebungsvariablen `$cname` und `$year` an die Werte »Brussels« und 1995 gebunden. Dann ergibt die Auswertung des XPath-Ausdrucks

```
//city[name=$cname]/population[@year=$year]
```

das Element

```
<population year="1995">951580</population>.
```
□

Seit XPath 2.0 können Variablen auch lokal, entweder durch Quantoren in *StepQualifiers* oder in Programmierkonstrukten gebunden werden. Mit Prädikaten der Form

```
[every|some $var in expr1 satisfies expr2]
```

können mit *StepQualifiers* sehr viel komplexere Bedingungen als in XPath 1.0 ausgedrückt werden.

Der Ausdruck `//country[every $c in city satisfies $c/population > 100000]/name` selektiert zum Beispiel die Namen aller Länder, in denen alle eingetragenen Städte mehr als 100.000 Einwohner haben.

Zusätzliche – mit XPath 2.0 eingeführte – iterative und konditionale Ausdrücke der Formen

```
for $var in expr return expr
```
und
```
if (expr) then expr else expr
```

erweitern XPath von einem reinen navigationsbasierten Adressierungsmechanismus zu einer eingeschränkten Programmiersprache. In Anbetracht der Tatsache, dass zu dieser Zeit solche Funktionalität bereits (besser) durch XQuery (siehe Abschnitt 3.5) abgedeckt wurde, erscheinen diese Erweiterungen etwas fragwürdig.

Funktionsbibliothek zu XPath und XQuery

Eine Menge nützlicher Funktionen und Operatoren wurde in *W3C XQuery 1.0 and XPath 2.0 Functions and Operators* [XQPFO01] zusammengefasst. Diese Operatoren – in XPath und XQuery zu benutzen – beinhalten Konvertierung, Berechnungen mit numerischen Werten (einschließlich Aggregation), Strings, Zeit und Datum sowie Dauer, wie bereits aus SQL bekannt. Zusätzlich werden bereitgestellt: XML-spezifische Operatoren wie z.B. Oberflächengleichheit und Tiefengleichheit, flaches und tiefes Kopieren von Knoten, Operationen auf Folgen von Knoten (z.B. Abbildung von Folgen von `IDREF`s auf die entsprechende Folge von referenzierten Knoten, oder Abbildung einer ID auf eine Folge der Knoten, die diese ID referenzieren), Kontextfunktionen und Vergleiche bzgl. Dokumentordnung. Die Funktion `document(`*`url`*`)`, welche das Document mit der *url* selektiert, ist ebenfalls hier definiert.

Eine erste formale Semantik für XPath – damals noch XSL Patterns bzw. XPath 1.0 – wurde in [Wadl99a, Wadl99b] angegeben. Das W3C spezifizierte die formale Semantik dann in 2000 in dem *W3C XML Query Data Model* (später

W3C XQuery 1.0 and XPath 2.0 Data Model [XQPDM01]; siehe Abschnitt 3.4.2) und der *W3C XML Query Algebra* (später *W3C XML Query Formal Semantics* [XMLQF]; siehe Abschnitt 3.4.3).

XPath als Grundkonzept für Sprachen in der XML-Welt

XPath ist keine vollständige XML-Anfragesprache: Im Vergleich mit der relationalen Algebra fehlt insbesondere ein Join-Operator (Semi-Joins werden durch die *StepQualifiers* ermöglicht). XPath kann auch keine beliebigen XML-Bäume *erzeugen*. Sein Zweck ist, als allgemeiner *Adressierungsmechanismus* für XML als Basis für XML-Anfragesprachen und weitere Konzepte zu dienen.

Ein wichtiger Aspekt hierbei ist das Design der XML-Welt als einer Menge von Standards (Anfragesprache, Metadatenbeschreibungssprache, Web-orientierte Funktionalität sowie weitere, anwendungsspezifische Konzepte), die allesamt auf XPath als Adressierungsmechanismus aufbauen und ihrerseits in XML-Syntax formuliert sind. Dieses Bild wird mittlerweile z.B. durch XSLT als Transformationssprache, XQueryX als Syntax für XQuery in XML, XML Schema zur Beschreibung von Schemainformation, und XLink zur Spezifikation von Verknüpfungen zwischen Dokumenten komplettiert.

3.3 Die Entwicklung zu XQuery

3.3.1 XQL

XQL (*XML Query Language*) [XQL98, XQL99] ist ein früher Vorschlag für eine einfache XML-Anfragesprache. Die zugrunde liegende Idee und Syntax – die Verwendung von Pfaden sowie Bedingungen zur Navigation – war dieselbe wie in XSL Patterns; dennoch waren die später für XPath zentralen Begriffe *Achse* und *location path* noch nicht definiert. XQL verwendete nur die Operatoren »/«, »//« und »/@«, um zu direkten Subelementen, transitiven Subelementen und Attributen zu navigieren – grob gesehen war es das Fragment von XPath (mit der abgekürzten Notation), das ohne Verwendung von `axis::` ausgedrückt werden konnte. Außerdem waren `union` und `intersect` von Ergebnismengen erlaubt.

Die zentrale Erweiterung, dank derer XQL nicht nur als Adressierungsmechanismus, sondern auch als Anfragesprache gelten konnte, war die Möglichkeit, einen Ergebnisbaum als Projektion und sogar als eingeschränkte Restrukturierung eines XML-Dokuments zu erzeugen. Dazu definierte der erste Vorschlag [XQL98] *Return-Operatoren*, die für jedes Präfix eines XQL-Pfadausdrucks Teile des gerade behandelten Knotens zu dem Ergebnisbaum beitragen konnten. Der spätere Vorschlag [XQL99] ersetzte diese Return-Operatoren dann durch *Gruppierung*, wobei (korrelierte) Subanfragen geschachtelte Ergebnisse berechnen konnten. Derselbe Vorschlag enthielt außerdem Funktionalität, um die Ordnung von Elementen abzufragen und für das Ergebnis zu ändern.

Weiterhin wurden *asymmetrische Joins* durch die Kombination von Korrelationsvariablen und Return-Operatoren unterstützt: Variablenbindungen der

Form *path*[$*var* := *expr*] in äußeren Schritten konnten weiter innen als Joinvariablen in absoluten Pfadausdrücken verwendet werden, außerdem konnten Teile der Elemente mit Hilfe der Return-Operatoren in den Ergebnisbaum eingefügt werden.

Da auch IDREF(S)-Attribute über Joins behandelt werden mussten, war die Verarbeitung von IDREFS-Attributen problematisch: Hier ist der Attributwert eine Folge von IDs, z.B. bei memberships="org-EU org-UN org-NATO ...". Eine Equijoin-Verknüpfung dieses Wertes z.B. mit <organization id="org-EU"> versagt. Diese Funktionalität musste nicht-deklarativ durch Stringoperationen simuliert werden.

Die beschriebene Zusatzfunktionalität, d.h. Return-Operatoren, Gruppierung und Umordnung, wurde nicht für XPath übernommen, da XPath 1.0 als reiner Adressierungsmechanismus entworfen wurde. Stattdessen wird entsprechende Funktionalität von den Transformations- und Anfragesprachen XSLT, XML-QL und Quilt/XQuery angeboten.

XQL beeinflusste das Design von XPath und weiterer Konzepte. Man beachte, dass XPath 1.0 weder Variablenbindungen, noch geschachtelte Subanfragen enthielt, die erst in XPath 2.0 hinzukamen. XQL wurde in [HuMa99] implementiert. Es diente in frühen Versionen von Tamino [Tamino] und Excelon [Excelon] als Anfragesprache, wo es inzwischen von XPath ersetzt wurde.

3.3.2 XML-QL

XML-QL [DFFL98a, DFFL99] ist ein anderer früher (1998, nicht-W3C) Vorschlag für eine XML-Anfrage- und -Transformationssprache. Das Design (und die Implementierung) von XML-QL wurde stark von dem Strudel/StruQL-Projekt beeinflusst [FFLS97, FFKL98]. Im Gegensatz zu *XSL Patterns* und XQL verwendet XML-QL keine Navigation und Pfade, sondern XML-Patterns, die mit dem XML-Dokument zur Überdeckung gebracht werden und damit Variablenbindungen erzeugen. Die zugrunde liegende Idee, XML-QL-Anfragen in einen *selection part* (WHERE ... IN) und einen *construction part* (CONSTRUCT) zu unterteilen, die über Variablenbindungen kommunizieren, war von SQL-artigen Sprachen motiviert:

WHERE *xml-pattern*$_1$
IN *url*
CONSTRUCT *xml-pattern*$_2$

Dabei wird *xml-pattern*$_1$ mit der unter *url* zugreifbaren XML-Instanz zur Überdeckung gebracht. Jede Überdeckung ergibt Bindungen für die in dem Pattern enthaltenen Variablen, die dabei auch gleichzeitig – wie in Datalog – als implizite Join-Variablen dienen. Mit jeder dieser Variablenbindungen wird *xml-pattern*$_2$ instanziiert und erzeugt somit das Ergebnisdokument.

Die folgende XML-QL-Anfrage transformiert die *name*-Subelemente der *country*-Elemente in Attribute. Die WHERE-Klausel passt auf alle *country*-Elemente in dem Dokument »www.../mondial.xml«, wobei die Variablen $id und $name an

den Wert des *car_code*-Attributs und den Textinhalt des *name*-Elements gebunden werden:

```
WHERE <country car_code=$id>
         <name>$name</>
       </>
IN "www.../mondial.xml"
CONSTRUCT <country car_code=$id name=$name> </>
```

Für jede Variablenbindung erzeugt die CONSTRUCT-Klausel ein neues *country*-Element im Ergebnis:

```
<result>
  <country car_code="D" name="Germany"> </>
  <country car_code="F" name="France"> </>
  :
</>
```

Die WHERE-Klausel kann auf mehrere Patterns erweitert werden:

WHERE (*xml-pattern*$_1$ IN *expr*$_1$, ... , *xml-pattern*$_n$ IN *expr*$_n$)

wobei *expr*$_i$ entweder ein Dokument, eine Variable die durch eine vorhergehende Klausel gebunden wurde, oder eine geschachtelte WHERE-CONSTRUCT-Klausel ist.

Die folgende XML-QL-Anfrage selektiert alle Elemente, die ein direktes *name*-Subelement mit Textinhalt »Monaco« besitzen:

```
WHERE $element IN mondial.xml,
      <name>Monaco</> IN $element
CONSTRUCT $element
```

Analog zu SQL kann die CONSTRUCT-Klausel geschachtelte XML-QL-Anfragen enthalten, womit z.B. der Inhalt von Elementen restrukturiert werden kann.

Dereferenzierung wird in XML-QL nicht direkt unterstützt. Das Problem wird gelöst, indem Referenzattribute wie Elemente behandelt werden, d.h., das XML-QL-Pattern

```
WHERE <country> <name> $cname </>
                <capital> <name> $capname </> </> </>
IN mondial.xml
CONSTRUCT <result name=$cname capital=$capname> </>
```

hat dieselbe Semantik wie XPath's `//country/@capital=>city/name`. Falls es außerdem *capital*-Subelemente gäbe, wäre die Semantik von XML-QL nicht wohldefiniert. Diese Lösung ist also zumindest fragwürdig. Da XML-QL aber Joins unterstützt, ist die äquivalente Formulierung in der folgenden Form vorzuziehen:

```
WHERE <country capital = $cap> <name> $cname</> </> IN mondial.xml,
      <city id = $cap> <name> $capname</> </> IN mondial.xml
CONSTRUCT <result name=$cname capital=$capname> </>
```

Wie bereits für XQL besprochen, ist auch die Behandlung von IDREFS-Attributen in XML-QL unzureichend, da Referenzen wie oben beschrieben durch Joins oder geschachtelte Subanfragen (also de facto auch Joins) ausgewertet werden müssen. Für die Aufspaltung von IDREFS sind auch hier nicht-deklarative Stringoperationen notwendig. In Anbetracht der Tatsache, dass dieselbe Veröffentlichung außer der Sprache XML-QL auch ein *graphbasiertes* Datenmodell vorstellt, welches den (dort an den Kanten markierten) XML-Baum durch *cross edges* zur Repräsentation von Referenzattributen erweitert, ist es überraschend, dass die sich daraus ergebenden Konsequenzen beim Design von XML-QL nicht berücksichtigt wurden.

Weitere Features in XML-QL beinhalten *Tag-Variablen* und *regular path expressions*, ähnlich wie in Lorel [GMPQ97, AQMW97] definiert. Im Vergleich zu XQL bietet XML-QL sehr viel mehr Anfrage- und Transformationsfunktionalität. Im Vergleich zu XSLT jedoch ist die Funktionalität als Transformationssprache deutlich eingeschränkt. Komplexe Transformationen sind nur kompliziert durch tief geschachtelte iterative WHERE-Klauseln und komplexe CONSTRUCT-Klauseln ausdrückbar. Wie auch bei SQL werden rekursive Anfragen von XML-QL nicht unterstützt. XML-QL wurde in [DFFL98b] implementiert.

3.4 Anfragen an XML: Theoretischer Hintergrund

Aufbauend auf den Erfahrungen mit XSL Patterns, XQL, XSLT und XML-QL wurden die Anforderungen an XML-Anfragesprachen in den *W3C XML Query Requirements* [XMLQR01] fixiert. Das *W3C XQuery 1.0 and XPath 2.0 Data Model* [XQPDM01] (früher *XML Query Data Model*) und die *W3C XQuery Formal Semantics* [XQFS01] (früher *XML Query Algebra*) bilden die formale Grundlage für XML-Anfragen, indem Typen und Operatoren definiert werden. Die Semantik von XPath und – später – XQuery wird seitdem bezüglich dieser Dokumente definiert.

3.4.1 Anforderungen an XML-Anfragesprachen

Die folgenden Anforderungen an die zukünftige XML-Anfragesprache wurden in den *W3C XML Query Requirements* [XMLQR01] festgelegt:

- Sie muss deklarativ sein und darf keine bestimmte Evaluierungsstrategie zusätzlich zu der explizit definierten Semantik erfordern.
- Sie muss die in XML 1.0 definierten Datentypen sowie die in XML Schema definierten *complexTypes* unterstützen.
- Sie muss Referenzen sowohl innerhalb eines Dokuments als auch zwischen Dokumenten (vgl. die Sprache *W3C XLink*) unterstützen.
- Sie muss mehrere Sprachbindungen besitzen. Eine Sprachsyntax muss für Benutzer verständlich sein, und eine Syntax muss in XML ausgedrückt werden. Die zugrunde liegende Idee ist dabei – wie auch bereits in XSLT –, dass

Anfragen selber XML-Instanzen sind und damit auch in den entsprechenden Sprachen manipuliert werden können.

3.4.2 Das Datenmodell für XML-Anfragesprachen

Das *XQuery 1.0 and XPath 2.0 Data Model* (früher *XML Query Data Model*) definiert auf formale Weise die Informationsstruktur der Eingabe für einen XML-Anfrageprozessor. Die Eingabe wird dabei als Baum mit markierten Knoten und Knotenidentität dargestellt. Diese Datenstruktur definiert alle in Berechnungen von XPath, XSLT sowie XML-Anfragesprachen erlaubten Werte. Sie verfeinert und formalisiert das abstrakte XML-Datenmodell (wie in *W3C XML Information Set* [XMLInf01] definiert und in Kapitel 1 dieses Buches beschrieben). Das grundlegende Konzept des Datenmodells sind *Knoten*, wobei sieben Knotentypen unterschieden werden können: *Dokument*, *Element*, *Attribut*, *Text*, *Namensraum*, *Verarbeitungsanweisung* (*processing instruction*) und *Kommentar*. Für jeden Knotentyp sind *Zugriffsoperatoren* (*accessors*) definiert, die Informationen über den Knoten zurückgeben. Einige werden im Folgenden beschrieben:

- *Dokument*: Ein Dokumentknoten wird durch seine URI-Referenz identifiziert. Er besitzt eine nicht leere Folge von Kindknoten, wobei genau einer davon ein Elementknoten sein muss.
- *Element*: Ein Elementknoten besitzt ein Tag und besteht aus einer ungeordneten Menge von Namensraumknoten, einer ungeordneten Menge von Attributknoten, und einer geordneten Folge von Kindknoten, die wiederum Elementknoten, Textknoten, Verarbeitungsanweisungen und Kommentare sein können.

Die Zugriffsoperatoren *name*, *namespaces*, *attributes*, *children*, *type* und *nodes* geben die entsprechenden Bestandteile des Elementknotens zurück (*nodes* gibt eine geordnete Folge der Attribute, gefolgt von den Subelementen zurück). Der Zugriffsoperator *parent* gibt den (eindeutigen) Elternknoten im XML-Baum zurück.

- *Attribut*: Attributknoten haben einen Namen und einen Wert, wobei der Datentyp des Wertes ein *simpleType* wie in XML Schema (Datatypes) ist.
- *Text*: Textknoten enthalten die tatsächliche Information. Ihr Typ ist ein primitiver XML-Schema-Datentyp. Mit den Zugriffsoperatoren *type*, *string* und *parent* lässt sich die jeweilige Information über Textknoten abfragen.

Neben Knoten unterstützt das Datenmodell (geordnete) Folgen und (ungeordnete) Multimengen von Knoten, welche z.B. die formale Grundlage für *Kontexte* bei der Auswertung von XPath-Schritten sowie Ergebnismengen von XPath-Ausdrücken darstellen.

Aus dem Datenmodell ergeben sich mehrere Einschränkungen und Anforderungen an die Anwendungen (d.h. hauptsächlich Anfragesprachen), die dieses Datenmodell verwenden. In Abschnitt 3.6 wird aufgezeigt, dass die Eindeutigkeit

des Elternknotens eine deutliche Einschränkung bei Updates (und in noch höherem Maß bei Datenintegration) darstellt.

3.4.3 Formale Semantik von XML-Anfragesprachen

Zur formalen Semantik von XQuery wird eine Algebra vorgeschlagen. Der Vorschlag der *W3C XQuery Formal Semantics* [XQFS01] löst den älteren Vorschlag *W3C XML Query Algebra* ab. Es werden Datentypen und Operatoren auf diesen Datentypen definiert. Die Datentypen beschreiben die Struktur von Mengen von XML-Instanzen und Elementen – ähnlich wie bei DTDs oder XML-Schema-Spezifikationen. Die Operatoren definieren, wie ein Ergebnis aus gegebenen Operanden berechnet wird. Dabei wird eine Anfrage auf einen Operatorbaum abgebildet, der beschreibt, wie die Anfrage aus elementaren Operatoren aufgebaut ist (und entsprechend ausgewertet wird). Da jede Anfrage implizit auch eine Klasse von XML-Instanzen definiert, die als Antwort erzeugt werden können, kann für eine Anfrage ebenfalls ein zugehöriger Typ bestimmt werden. Basierend auf dem Operatorbaum definiert die Algebra die *Semantik* von Anfragen, indem sowohl das Ergebnis der Anfrage bezüglich einer gegebenen XML-Instanz als auch der *Ergebnistyp* der Anfrage abgeleitet wird: Die »statische Semantik« leitet den Ergebnistyp einer Anfrage durch strukturelle Induktion entsprechend den Typ-Ableitungsregeln ab. Die »dynamische Semantik« definiert das Ergebnis einer Anfrage an eine XML-Instanz mit Hilfe von Wert-Ableitungsregeln.

Die Datentypen sind eng mit den *complexTypes* von XML Schema verwandt. Während das Datenmodell und die Algebra als formale Grundlage der Typen und Operatoren dienen, wird mit XML Schema das Typsystem von XML spezifiziert. Die Algebra-Datentypen enthalten keine Entsprechung zu den `ID`- und `IDREF`-Attributtypen. Stattdessen werden Referenzen durch einen eigenen Datentyp sowie die Operatoren `ref` und `deref` repräsentiert.

Komplexe Datentypen werden durch (eingeschränkte) reguläre Ausdrücke definiert, wobei jeweils die Namen, Typen und Kardinalitäten angegeben werden. Weiterhin können die erlaubten Reihenfolgen der Subelemente spezifiziert werden durch: »,« für Sequenz; »&« für beides, in beliebiger Reihenfolge; »|« für Auswahl.

Beispiel – Datentypen. Die Typen der *mondial*- und *country*-Elemente der Mondial-Datenbank sind wie folgt spezifiziert:

```
type Mondial = ELEMENT mondial (((ELEMENT country)+) &
                                ((ELEMENT continent)+) &
                                ... )
type Country = ELEMENT country
    ( ATTRIBUTE car_code (xs:string) &
      ATTRIBUTE capital (REFERENCE (ELEMENT city (City))) &
      ATTRIBUTE memberships (REFERENCE (ELEMENT organization (Organization))*) &
      ATTRIBUTE area (xs:number),
      ELEMENT name (xs:string),
```

```
      ELEMENT population  (xs:number),
      ((ELEMENT indep_date  (xs:number))?),
         :
      ((ELEMENT ethnicgroups (CulturalInfo))*),
      ((ELEMENT religions (CulturalInfo))*),
      ((ELEMENT languages (CulturalInfo))*),
      ( (ELEMENT province (Province))* |
        (ELEMENT city (City))*)
    )
type CulturalInfo = ELEMENT culturalinfo (ATTRIBUTE percentage (xs:string) ,
                                          (xs:string))                    □
```

Für die Instanz-Ebene wird eine der XML-ASCII-Notation entsprechende Repräsentation verwendet.

Anfragen werden in der Algebra als Ausdrücke über Konstanten, Namen und Operatoren dargestellt, von denen einige im Folgenden beschrieben werden:

- Zugriffsoperatoren: Zur »Navigation« innerhalb der XML-Instanz dienen die vordefinierten Operatoren `children(`*expr*`)`, `attributes(`*expr*`)`, `parent(`*expr*`)`, `dereference(`*expr*`)`, `name(`*expr*`)`, `string-value(`*expr*`)` (und einige weitere).
- Iteration: Mit `for` *var* `in` *expr* `return` *expr* wird die Iteration über eine Folge von Knoten bezeichnet. Da es keinen abstrakten Join-Operator gibt, müssen Joins als *nested-loop-join* durch geschachtelte `for`-Iterationen beschrieben werden. Dabei ist zu beachten, dass for-Joins nicht kommutativ sind, da die Ordnung der Elemente in der Antwort relevant ist.
- Bedingungen: `if` *expr* `then` *expr* `else` *expr*,
- Lokale Variablenbindungen: `let` *var* `:=` *expr*,
- Vereinigung, Mengendifferenz, Schnittmengen,
- Arithmetische, boolesche und Gleichheitsoperatoren sowie Funktionsanwendungen,
- Sortierung: *expr* `sortby` *expr* `ascending|descending`,
- Konstruktoren für Elemente, Attribute und Referenzen.

Im Gegensatz zur relationalen Algebra, die aus den Operatoren *Selektion*, *Projektion*, *kartesisches Produkt/Join*, *Vereinigung* und *Mengendifferenz* sowie den abgeleiteten Operatoren *Schnittmenge* und *Division* besteht, bietet die *XQuery Formal Semantics* nur eine Menge sehr einfacher Operatoren, die etwa mit einer Spezifikation von SQL auf Implementierungsebene vergleichbar ist. Damit ist die *core* Algebra auch noch weit von den in XPath verwendeten Konzepten entfernt. Die XPath-Konstrukte werden als abgeleitete bzw. reduzierbare Ausdrücke der Algebra definiert.

Beispiel – Abbildung von XPath auf die Algebra. Der XPath-Ausdruck `document(...)/mondial/country` wird im Wesentlichen auf

```
[[document(...)/mondial/country]] = for $m in children(document(...)) return
                                      if name($m) = "mondial" then
```

```
                                        for $c in children($m) return
                                          if name($c) = "country" return $c
```

abgebildet. Ähnlich wird `//country/@capital=>city` übersetzt zu

```
for $c in [[//country]] return
  for $a in attributes($c) return
    if name($a) = "capital" then
      for $city in deref($a) return
        if name($city) = "city" then $city else ()
```

□

Derzeit wird daran gearbeitet, die verschiedenen durch das *Document Object Model* (DOM), *XML Schema*, das *XQuery 1.0 and XPath 2.0 Data Model* und die *W3C XQuery Formal Semantics* gegebenen »Sichten« auf das abstrakte XML-Datenmodell (wie in *W3C XML Information Set* [XMLInf01] definiert) miteinander in Einklang zu bringen.

3.5 XQuery

Die Anfragesprache *W3C XQuery* hat ihre Wurzeln in der 2000 vorgeschlagenen Sprache *Quilt* [ChRF00]. Quilt bettete XPath in höhere Konstrukte (ähnlich denen von SQL/OQL) ein, wie es zuvor bereits bei XML Schema und XLink für andere Zwecke geschehen war. Mit einigen kleinen Modifikationen wurde Quilt als W3C *XQuery* [XQuery01] (Feb. 2001 Working Draft, inzwischen die XPath-2.0-Semantik verwendend) übernommen. Da die meisten Designentscheidungen bereits bei der Entwicklung von Quilt getroffen wurden, mussten nur noch die »W3C-spezifischen« Aspekte der Integration von Quilt mit den neuesten Entwicklungen des *XML Query Data Model* sowie der *XML Query Algebra* gelöst werden. Seit dem Dezember 2001 Working Draft, beinhaltet die Definition von XQuery direkt die Definition von XPath 2.0.

XQuery-Anfragen bestehen aus einer Folge von Klauseln, die deklarativ beschreiben,

- welche Information verwendet werden soll,
- welche weiteren Bedingungen erfüllt sein müssen, und
- wie daraus das Ergebnis erzeugt wird.

Die Schlüsselwort dieser XQuery-Klauseln sind `FOR - LET - WHERE - RETURN` (kurz FLWR oder »flower«):

```
FOR variable1 IN xpath-expr1
LET variable2 := xpath-expr2
WHERE conditions
RETURN xml-expr
```

Diese Grundstruktur kann erweitert werden, indem jede `FOR`-Klausel und jede `LET`-Klausel mehrere Variablen binden kann. Variablen, die in einer `FOR`- oder `LET`-Klausel gebunden sind, können in nachfolgenden `IN`-Klauseln verwendet werden.

Die FOR-Klauseln definieren Variablen, bei denen über die einzelnen Elemente der Ergebnismengen von XPath-Ausdrücken iteriert wird. Die Variablen in der LET-Klausel werden an das komplette Ergebnis des entsprechenden XPath-Ausdrucks gebunden, enthalten also im Allgemeinen eine Folge von Knoten. Solche Variablen, die in einer LET-Klausel gebunden sind, können – neben der Übernahme einer ganzen Gruppe von Knoten in das Ergebnis – auch für Gruppierungen und Aggregierungsfunktionen verwendet werden. Das Ergebnis der FOR- und LET-Klauseln ist damit eine Folge von Tupeln von Variablenbindungen.

Die WHERE-Klausel drückt – wie in SQL/OQL – weitere Bedingungen aus, welche Tupel von Variablenbindungen zur Erzeugung des Ergebnisses weiter betrachtet werden sollen. Hierbei wird dieselbe Syntax wie für Bedingungen in *StepQualifiers* in XPath verwendet. Schlussendlich erzeugt die RETURN-Klausel einen XML-Teilbaum für jedes Tupel von Variablenbindungen.

Beispiel. Die im Folgenden angegebene XQuery-Anfrage erzeugt eine XML-Instanz, die aus allen Ländern besteht, die mehr als 1.000.000 Einwohner haben und mindestens 10 Städte (als Subelemente) enthalten:

```
FOR $c IN document("mondial.xml")//country
LET $cities := $c//city
WHERE $c/@population > 1000000 und count($cities) > 10
RETURN
  <bigcountry population = {$c/@population}>
    <name> {$c/@name}</name>
    {$cities}
  </bigcountry>
```

Hierbei läuft $c über alle *country*-Elemente. Für jeden Wert von $c wird die Variable $cities an die Folge aller *city*-Subelemente von $c gebunden, auf die dann zuerst eine Aggregierungsfunktion angewendet wird, und am Ende die gesamte Folge in das Ergebnis übernommen wird. □

Wie bereits in SQL und in aller Konsequenz und Eleganz in OQL, können FLWR-Klauseln in FOR - IN (FLWR), WHERE ... (FLWR) ... und RETURN ... (FLWR) ...-Klauseln geschachtelt werden – kurz gesagt, überall wo ein beliebiger Ausdruck erwartet wird, kann auch ein FLWR-Ausdruck stehen.

Wie der Name schon sagt – ein Quilt ist ein Flickenteppich – ist Quilt/XQuery ein Patchwork, dessen Design von vielen Anfragesprachen beeinflusst wurde:

- SQL/OQL: Anfragen bestehen aus einer Folge von Klauseln.
- OQL: eine funktionale Sprache, in der Teilausdrücke beliebig geschachtelt sein können.
- XPath/XQL/XSL Patterns: XPath-Ausdrücke werden zur Navigation in XML-Bäumen verwendet.
- XQL: ein FILTER-Konstrukt ermöglicht die Projektion von Baumfragmenten, ähnlich den Return- und Gruppierungsoperatoren in XQL.
- XML-QL:

- Die Struktur von XQuery-Anfragen als Ganzes ist ähnlich derjenigen von XML-QL (`WHERE` *xml-pattern* `IN` *url* `CONSTRUCT` *xml-pattern*).
- Information wird mit Variablenbindungen von dem Extraktions-Fragment (`FOR - LET - WHERE`) zu dem Erzeugungs-Fragment (`RETURN`) kommuniziert.
- In beiden Fällen wird das Ergebnis durch Instanziierung eines XML-Patterns mit Variablenbindungen erzeugt (wobei optional Teilbäume und Gruppierungen durch geschachtelte FLWR-Ausdrücke berechnet werden können).
- Im Unterschied zu XML-QL, wo im Extraktions-Fragment ein XML-Pattern, das mit der Eingabeinstanz überdeckt wird, verwendet wird, basiert XQuery auf Iteratoren und XPath-Ausdrücken.

Mit XQuery lassen sich die in Datenbankanfragen »üblichen« Operatoren ausdrücken:

Joins

In XQuery werden *Joins* in der `FOR`-Klausel durch die Angabe mehrerer »*var* `IN` *xpath-expr*«-Definitionen oder durch eine Folge von `FOR`- und `LET`-Klauseln ausgedrückt. Jede `FOR`- oder `LET`-Klausel darf dabei Referenzen auf in vorhergehenden Klauseln definierte Variablen enthalten. Dieses ermöglicht »korrelierte Joins«, bei denen die zweite Komponente bereits von dem gerade betrachteten Wert der ersten Komponente abhängt (wie auch in OQL und neueren SQL-Implementierungen möglich).

Beispiel – Joins in XQuery. Die folgende Anfrage bestimmt alle Paare von Ländern und Städten, die denselben Namen haben, und gibt den Namen und Landescode des Landes sowie den Landescode der Stadt aus:

```
FOR $country in //country, $city in //city
WHERE $country/name/text() = $city/name/text()
RETURN <pair name={$country/name/text()}
             country={$country/@car_code}
             citycountry={$city/@country}/>
```

Eine äquivalente Formulierung mit einem korrelierten Join ist

```
FOR $country in //country, $city in //city[name/text() = $country/name/text()]
RETURN <pair name={$country/name/text()}
             country={$country/@car_code}
             citycountry={$city/@country}/>
```

(die in etwa einer SQL-Anfrage mit einer SFW-Klausel in der `FROM`-Zeile entspricht). □

Weitere Join-artige Funktionalität kann durch geschachtelte FLWR-Ausdrücke in der `RETURN`-Klausel erreicht werden. Auch hier kann der innere FLWR-Ausdruck auf Variablen der Umgebung zugreifen.

Geschachtelte FLWR-Ausdrücke in der WHERE-Klausel bieten nur eine auf die Auswertung von Bedingungen eingeschränkte Semi-Join-Funktionalität, da die in der inneren Anfrage berechneten Werte nicht an die Berechnung des Ergebnisses weitergegeben werden können (dieselbe Situation ist auch in SQL/OQL gegeben).

Selektion

Selektionsbedingungen werden in XQuery (wie in SQL/OQL) explizit in der WHERE-Klausel untergebracht, zusätzlich können Selektionsbedingungen in den XPath-Ausdrücken in den FOR- und LET-Klauseln (*StepQualifiers*) untergebracht werden. Zusätzlich zu den üblichen Bedingungen können `precedes` und `follows` als Prädikate verwendet werden, um Bedingungen zu formulieren, die sich auf die Dokumentordnung beziehen. Eng mit selektiver Funktionalität verwandt ist außerdem das IF-THEN-Konstrukt in XPath 2.0, das hier auch eine sinnvolle Spracherweiterung ist.

Projektion

Projektion wird im Wesentlichen durch die Definition von Variablen in den FOR - LET-Klauseln sowie einen expliziten auf XPath-Ausdrücken basierenden FILTER-Operator bereitgestellt: Der Ausdruck

xpath-expr$_1$ FILTER *xpath-expr*$_2$

erzeugt einen Baum, der die Ergebnismenge *xpath-expr*$_1$ auf diejenigen Knoten reduziert, die (bezüglich der Eingabe) von *xpath-expr*$_2$ selektiert werden. Damit können aus einem selektierten Teilbaum noch Knoten bzw. Ebenen *entfernt* werden. Jeder Knoten wird jeweils *ohne* Attribute oder Subelemente betrachtet, d.h., ein selektierter Knoten trägt primär nur seine Tags oder – falls er ein Textknoten ist – seinen Text bei. Alle Attributknoten und Elementknoten, die zum Ergebnis beitragen sollen, müssen explizit eigenständig den Filter erfüllen. Damit kann der FILTER-Operator die Baumstruktur tiefgreifend verändern: Elemente, die ursprünglich indirekte Subelemente waren, können zu direkten Subelementen werden, wenn die dazwischenliegenden Knoten in der Projektion ausgefiltert werden.

Restrukturierung/Ergebniserzeugung

Eine sehr flexible Restrukturierungsfunktionalität ergibt sich zum einen aus der Verwendung von XPath-Ausdrücken in den FOR- und LET-Klausel, die eine integrierte Kombination von Selektion und Projektion erlauben. Die konstruktive Funktionalität ist in der RETURN-Klausel angesiedelt, wo neue XML-Teilbäume aus den selektierten Fragmenten zusammengesetzt werden. Zusätzlich zu der Erzeugung durch XML-Patterns sind *computed Element/Attribute constructors* wie in der Algebra-Notation erlaubt, um Elemente und Attribute zu erzeugen, deren Namen als Variableninhalte berechnet werden (was ansonsten z.B. durch Tag-Variablen geschehen würde).

Beispiel. Das folgende Fragment erzeugt ein Element durch Verwendung expliziter Konstruktoren:

```
element {$tag} {attribute {$attrname} {$attrval}
                element {$elemname} {$contents}
                }
```
□

Innerhalb der RETURN-Klausel wird durch geklammerte Ausdrücke der Form {*expr*} angegeben, dass der Ausdruck ausgewertet und sein Ergebnis in die zu erzeugende Struktur eingesetzt werden soll (wie weiter oben bereits mehrmals verwendet), während ohne die Klammerung der Ausdruck selber als Text eingesetzt würde.

Weitere Operatoren

Einige Operatoren, die bereits aus SQL/OQL bekannt sind, sind ebenfalls erlaubt: Die FOR-Klausel kann durch eine DISTINCT-Anweisung erweitert werden. Aggregierungsfunktionen, z.B. max(*expr*) wie in SQL/OQL, sind überall erlaubt, wo Ausdrücke erwartet werden. Eine SORTBY *expr*-Klausel kann an die RETURN-Klausel angehängt werden, um die erzeugten Knoten zu sortieren. Die aus SQL/OQL bekannten Quantoren ANY und ALL haben in XQuery eine explizitere, mathematischere Syntax der Form

```
WHERE SOME/EVERY var IN expr SATISFIES predicate .
```

Beispiel. Die folgende Anfrage ergibt für jedes Land, das mindestens eine Stadt mit mehr als 1.000.000 Einwohnern enthält, ein Element, das den Namen des Landes sowie seine gesamte städtische Bevölkerung ausgibt:

```
FOR $c IN //country
WHERE SOME $city IN $c//city SATISFIES $city/population > 1.000.000
RETURN <result name = {$c/name/text()}
              urban = {sum($c//city/population)}/>
```
□

Wie in XML-QL (und SQL) sind rekursive Anfragen, z.B. zur Berechnung einer transitiven Hülle, mit den reinen XQuery-Befehlen nicht möglich. Die Verwendung benutzerdefinierter Funktionen – ähnlich wie mit PL/SQL – macht XQuery dann Turing-vollständig.

XQueryX

Als *W3C XQueryX* [XQX01] wurde – wie in *XML Query Requirements* (vgl. Abschnitt 3.4.1) gefordert – eine XML-Syntax für XQuery definiert.

Offene Fragen

Ein wichtiger Aspekt bezüglich der Implementierung und Verwendung von XQuery ist, dass Selektionsfunktionalität sowohl durch die eingebetteten XPath-Ausdrücke als auch in der WHERE-Klausel bereitgestellt wird (vgl. Beispiel 3.13). Dieses Problem ist ähnlich der Verwendung von geschachtelten SELECT-Klauseln in

SQL in der FROM-Klausel, um die Eingabemenge für Joins frühzeitig zu reduzieren. In SQL-Implementierungen wird dazu eine interne (theoretisch Algebra-basierte) Optimierung vorgenommen, bei der der Operatorbaum transformiert wird. Für XQuery ist dies eine offensichtliche Anwendung für die *XQuery Formal Semantics* (siehe Abschnitt 3.4.3).

Bezüglich der in den *XML Query Requirements* festgelegten Anforderungen ist die Behandlung von Inter-Dokument-Referenzen bis jetzt nicht abgedeckt. Das Problem wird in [May02] untersucht und ein möglicher Lösungsvorschlag beschrieben.

3.6 Verändern von XML-Daten

Die vorgestellten XML-Anfragesprachen (Stand Mitte 2002) bieten keine Konstrukte, um XML-Daten zu verändern. Der Sprachentwurf von XQuery 1.0 enthält keine entsprechenden Befehle, aber es ist vorgesehen, dass diese in späteren Versionen ergänzt werden. Somit können Anwendungsprogramme XML-Daten bisher nur durch expliziten Zugriff auf interne Datenstrukturen (z.B. als DOM) aus Host-Programmiersprachen verändern.

[TIHW01] beschreibt einen Vorschlag zur Erweiterung von XML-Anfragesprachen zu XML-Datenmanipulationssprachen. Es wird eine sprachunabhängige Definition angegeben, unter der Annahme, dass die Veränderungen in einer auf Variablenbindungen basierenden Sprachumgebung (wie z.B. in XML-QL oder XQuery) formuliert werden. Das *Target* eines Modifikationsbefehls ist der Elementknoten, der verändert werden soll. Die folgenden Operationen können auf das *Target* angewendet werden:

- `Delete(`*`member`*`)`: Wenn `member` ein Subelement (einschließlich Textelementen), ein Attribut oder eine Referenz innerhalb eines `IDREF(S)`-Attributs ist, wird es gelöscht.
  Insbesondere ist es möglich, einzelne Referenzen innerhalb eines `IDREFS`-Attributs, z.B. die Mitgliedschaft Belgiens in der EU, zu löschen. In diesen Fällen ist es jedoch nicht möglich, eine solche Referenz durch XPath deklarativ auszuwählen.
- `Rename(`*`member`*`,`*`name`*`)`: Wenn *`member`* ein Subelement (kein Text-Subelement) oder ein Attribut ist, wird es umbenannt. Für `IDREFS` kann nur die gesamte Liste umbenannt werden, eine Umbenennung einzelner Referenzen ist nicht möglich.
- `Insert(`*`content`*`)`: Hier kann *`content`* Text, ein Element, ein Attribut oder eine Referenz sein. Wie weiter unten detailliert beschrieben wird, sind Einfügungen unter der Einschränkung des XML Query Data Model, dass jedes Element einen eindeutigen Elternknoten haben muss, kritisch.
- `InsertBefore(`*`ref`*`,`*`content`*`)`: *`ref`* und *`content`* müssen entweder beides Elemente oder Text oder beides Referenzen sein. In diesem Fall wird *`content`* vor *`ref`* eingefügt. Entsprechend für `InsertAfter(`*`ref`*`,`*`content`*`)`.

- `Replace(`*`member`*`,`*`content`*`)` ist eine atomare Ersetzungsoperation, die äquivalent zu `InsertBefore(`*`member`*`,`*`content`*`)` gefolgt von `Delete(`*`member`*`)` ist.

Basierend auf diesen abstrakten Operationen wird beschrieben, wie die XQuery-Syntax zu `FOR-LET-WHERE-UPDATE`-Klauseln erweitert werden kann. QuiP [Quip01] implementierte eine vergleichbare Erweiterung von XQuery um Update-Operationen, wobei eine andere Syntax verwendet wird [Leht01].

Kritik

Wenn man dem Ansatz das *XML Query Data Model* zugrunde legt, welches fordert, dass jedes Element einen eindeutigen Elternknoten haben muss, ergeben sich Probleme für Einfügungen und Ersetzungen: Wenn eine Variable in *`content`* an einen Teilbaum aus dem Originaldokument gebunden ist, muss dieser Teilbaum zur Einfügung kopiert werden. Damit ist dann nicht offensichtlich, ob Referenzen in den ursprünglichen Teilbaum weiterhin in das Original oder in die Kopie zeigen sollen. Eine detaillierte Analyse wird in [MaBe01] beschrieben.

3.7 Präsentation und Transformation von XML-Daten

Wie bereits für SGML-Daten werden zur Präsentation von XML-Daten *Stylesheets* verwendet. Dabei wird für jedes (darzustellende) XML-Dokument ein Stylesheet angegeben, mit dem das Dokument nach HTML transformiert wird, um dann in einem Browser darstellbar zu sein. Als Transformationssprache wird im XML-Bereich die Sprache XSL/XSLT verwendet. In der Anfangsphase der Entwicklung bestand XSL aus drei Komponenten:

- *XSL Patterns* als Adressierungs- und Selektionsmechanismus innerhalb von XML-Dokumenten. Dieser Teil wurde 1999 als *XPath* separiert.
- *XSL-FO* (Formatting Objects; innerhalb des Namensraums »`fo`«) bietet Elemente, die die Layout-Formatierung beschreiben, etwa vergleichbar den Formatierungsumgebungen in LaTeX, z.B.
  - Seitenlayout, Bereiche, Rahmen, Einrückung,
  - Farben, Schriftarten und -größen,
  - optisches Markup als Listen, Tabellen usw.

  Solche Elemente werden bei der Transformation (siehe unten) in das Ergebnisdokument eingefügt. XSL-FO-fähige Browser können diese Elemente direkt gemäß internen Vorschriften als HTML interpretieren und entsprechend darstellen. Andere Anwendungen können sie z.B. nach LaTeX/RTF/PDF usw. übersetzen.
- *XSLT* (XSL Transformations; innerhalb des Namensraumes »xsl«) [XSLT01] ist eine funktionale Programmiersprache, mit der XML-Dokumente transformiert werden können. Diese wird im Folgenden etwas genauer beschrieben. Einen tiefen Einblick gibt z.B. [Kay00].

Im Gegensatz zu XQL und XML-QL, die als Anfragesprachen entworfen wurden, ist XSLT als Transformationssprache gedacht: XSLT-Stylesheets werden auf XML-Instanzen angewendet und ergeben eine neue XML-Instanz. Dabei ist ein XSLT-Stylesheet eine deklarative Spezifikation, wie Elemente (rekursiv) transformiert werden. Ein XSLT-Stylesheet ist selber auch eine gültige XML-Instanz (bzgl. der XSLT-DTD). Jedem der dabei verwendeten XSLT-Elementtypen ist eine spezielle Transformationssemantik zugeordnet, wobei zur Adressierung von Knoten im Quelldokument XPath-Ausdrücke als Attributwerte und Elementinhalt in diese Elemente eingebettet werden. Im Wesentlichen besteht ein solches Stylesheet aus Templates der Form

```
<xsl:template match="match-expr">
   contents
</xsl:template>
```

Dabei ist *match-expr* ein XPath-Ausdruck, der bezüglich des Wurzelelements des aktuell bearbeiteten Dokuments ausgewertet wird. Die Ergebnismenge gibt an, für welche Knoten das Template *anwendbar* ist. Die Entscheidung, für welche Knoten es bei einem gegebenen Programmablauf *tatsächlich angewendet* wird, geschieht an einer anderen Stelle. Auf diese Weise kann wie weiter unten beschrieben z.B. bei der Bearbeitung eines Elements angegeben werden, dass für jedes der Subelemente das zu ihm »passende« Template aufgerufen werden soll.

Der Inhalt des Templates, *contents*, ist eine Folge von XSLT-Elementen, die das »Programm« zur Transformation der ausgewählten Knoten bilden. Dieses fügt – u.a. durch rekursive Aufrufe weiterer Templates für Subelemente – Knoten und Textinhalt in den Ergebnisbaum ein. Neben dem direkten Einfügen von Werten können dabei auch Knoten und Werte aus dem Eingabebaum kopiert werden:

```
<xsl:template match="//country/province/city">
   <xsl:copy-of select="current()"/>
</xsl:template>
```

ist z.B. ein einfaches Template, das anwendbar ist auf *city*-Elemente, die Subelemente eines *province*-Elementes sind, das wiederum ein Subelement eines *country*-Elements ist. Es kopiert diese Knoten unverändert in den Ergebnisbaum.

Die tatsächliche Anwendung der Templates während der Ausführung wird durch `xsl:apply-templates`-Elemente der Form

```
<xsl:apply-templates select="xpath-expr"/>
```

(innerhalb des `contents` der `xsl:template`-Elemente) gesteuert: *xpath-expr* ist ein XPath-Ausdruck, der (bezüglich des gerade bearbeiteten Knotens als Kontextknoten) Elemente des Eingabebaumes selektiert, für die die passenden Templates angewendet werden.

- `<xsl:apply-templates select="city[population > 1000000]"/>`
  bearbeitet alle *city*-Subelemente des aktuellen Knotens, deren Textinhalt des *population*-Subelements > 1.000.000 ist. Man beachte, dass dieser Befehl

nicht spezifiziert, welches Template angewendet wird. Diese Auswahl geschieht – wie oben beschrieben – durch das `xsl:template match`-Attribut des *Template*-Elements (Strategien zur Konfliktlösung werden in [XSLT01] beschrieben).

- `<xsl:apply-templates select="id(@capital)"/>`
  bearbeitet das Element in dem aktuellen XML-Dokument, dessen ID mit dem Wert des *capital*-(Referenz)-Attributes des gerade bearbeiteten Knotens übereinstimmt.

Beispiel – XSLT Stylesheet. Das folgende XSLT-Stylesheet erzeugt einen Ergebnisbaum, der alle *country*-Elemente und danach alle *city*-Elemente enthält:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <!-- einfaches Template, das alle country- und city-Elemente in den
        Ergebnisbaum kopiert -->
   <xsl:template  match="city|country">
      <xsl:copy-of select="current()"/>
   </xsl:template>
   <!-- das Haupt-Template wird auf das Wurzelelement angewendet -->
   <xsl:template match="mondial">
      <!-- zuerst: alle country-Elemente mit dem (passenden) Template bearbeiten -->
      <xsl:apply-templates select="/mondial/country"/>
      <!-- ... und dann alle City-Elemente -->
      <xsl:apply-templates select="//country/city | //country/province/city"/>
   </xsl:template>
</xsl:stylesheet>
```
□

Weiterhin ist es möglich, Templates einen Namen zu geben und sie durch Elemente der Form

```
<xsl:call-template name="name"/>
```

an beliebigen Stellen explizit aufzurufen.

Erzeugung des Ergebnisbaums

Die Ausführung eines XSLT-Stylesheets bezüglich eines XML-Dokuments beginnt mit der Anwendung desjenigen Templates, das auf das Wurzelelement anwendbar ist. Die weitere Ausführung wird durch `<xsl:apply-templates>` und `<xsl:call-template>` gesteuert, wobei die einzelnen Templates zum Ergebnisbaum beitragen:

- Alle Tags, komplette Elemente und Attribute im *contents*-Teil eines ausgeführten Templates, die nicht im `xsl:`-Namensraum liegen (d.h. die keine XSLT-Befehle sind), werden in den Ergebnisbaum übernommen.
- Mit `<xsl:copy-of select="`*xpath-expr*`"/>` bzw. `<xsl:value-of select="`*xpath-expr*`"/>` werden Knoten bzw. Werte direkt vom Eingabebaum in den Ausgabebaum übernommen.

- Attributwerte können durch XPath-Ausdrücke berechnet und eingesetzt werden: `<elementname attribute="xpath-expr">`
- Neue Elemente können durch

  ```
  <xsl:element name="xpath-expr_1">
    <xsl:attribute name="xpath-expr_2">
       contents
    </xsl:attribute>
  </xsl:element>
  ```

  generiert werden. Dabei werden der Elementname sowie die Attributnamen durch $xpath\text{-}expr_i$ berechnet; die Erzeugung des Elementinhalts wird dann durch rekursive Anwendung der XSLT-Befehle in *contents* gesteuert.

Weiterhin bietet XSLT die bekannten prozeduralen Konzepte wie Schleifen und bedingte Verzweigungen zur Kontrolle des Verarbeitungsablaufs an. Mit

```
<xsl:for-each select="xpath-expr">
   contents
</xsl:for-each>
```

kann über die Ergebnismenge eines XPath-Ausdrucks iteriert (und die in *contents* angegebenen Befehle ausgeführt) werden. Bedingte Verzweigungen werden als Fallunterscheidungen ausgedrückt:

```
<xsl:if test="predicate"> contents </xsl:if>
```

oder

```
<xsl:choose>
   <xsl:when test="predicate_1"> contents_1 </xsl:when>
   <xsl:when test="predicate_2"> contents_2 </xsl:when>
   :
   <xsl:otherwise> contents_{n+1} </xsl:otherwise>
</xsl:choose>
```

XSLT-Variablen können definiert werden und *einmal* einen Wert (Teilbaum, Liste von Bäumen, Literal) zugewiesen bekommen:

```
<xsl:variable name="var-name" select="xpath-expr">
   contents
</xsl:variable>
```

wobei der Wert *entweder* durch das `select`-Attribut oder durch *contents* (womit Bäume erzeugt werden können) gegeben wird. Diese Variablen können dann in

```
<elementname attribute="$var-name"/>
```

zur Definition von Attributwerten oder in Elementen der Form

```
<xsl:value-of select="$var-name"/>
```

wobei der Wert der Variablen in einen String konvertiert wird, oder in

```
<xsl:copy-of select="$var-name"/>
```

womit der gesamte Variableninhalt – der wie oben beschrieben z.B. eine Liste von Teilbäumen sein kann – in den Ergebnisbaum eingefügt wird. Die Kommunikation von Variablen zwischen Templates wird durch *parametrisierte Templates* organisiert. Parameter für Templates können mit

```
<xsl:template match="...">
   <xsl:param name="param-name" select="xpath-expr"/>
</xsl:template>
```

deklariert werden. Wenn ein solches Template aufgerufen wird, wird der Parameterwert durch ein Attribut spezifiziert:

```
<xsl:apply-templates select="xpath-expr1">
   <xsl:with-param name="param-name" select="xpath-expr2"/>
</xsl:apply-templates>
```

XSLT ist – ebenso wie XML Schema (siehe Kapitel 2) und XQueryX – eine Sprache, die *selber* in XML formuliert ist. Dies ermöglich es, XSLT-Programme auch als XML-Daten zu sehen, auszutauschen und auch zu verarbeiten (z.B. in Programmtransformationen oder -analysen, ähnlich wie in LISP). Für einen tieferen Einblick in weitere Funktionalität und Programmiertricks sei an dieser Stelle noch einmal auf [Kay00] verwiesen.

3.8 Verwandte Arbeiten

Semistrukturierte Daten in der Zeit vor XML

Bereits vor der Definition von XML beschäftigten sich einige Projekte mit semistrukturierten Daten, z.B. Tsimmis/OEM/Lorel [GMPQ97, AQMW97], Strudel/StruQL [FFLS97, FFKL98] und F-Logic [KiLa89, KiLW95]. Diese Projekte und Sprachen beeinflussten die Entwicklung von XML (gemeinsam mit Einflüssen aus dem SGML-Bereich). Einige von ihnen wurden für XML fortgesetzt:

Lore [MAGQ97] wurde in [GoMW99] nach XML migriert. Das ursprüngliche Datenmodell wurde um die Unterscheidung zwischen Attributen und Subelementen erweitert. Lorel unterstützt nur die XML-Achsen *child* und *attribute*, bietet aber dafür weiterhin die regulären Pfadausdrücke der ursprünglichen Lorel-Sprache. Wie bereits beschrieben wurde, resultierte die Fortführung von Strudel/StruQL [FFLS97, FFKL98] für XML in der Sprache XML-QL. F-Logic [KiLa89] hatte mit seinem Frame-basierten Datenmodell und der auf Navigation und Spezifikationen (die den *StepQualifiers* in XPath entsprechen) basierenden Anfragesprache starke Ähnlichkeit mit XML/XPath. Der Ansatz wurde mit XPath-Logic und XPathLog fortgesetzt (siehe unten). YAT/YATL (*Yet Another Tree Model/Language*) [CDSS98] ist ein Prä-XML-Vorschlag, der bereits SGML und DTDs verwendete. Seine Bäume bieten ein gemeinsames Datenmodell für relationale, objektorientierte (ODMG) und semistrukturierte (SGML-)Daten. In

[ChCS00] wurde das YAT-System in ein System zur Datenintegration in XML weiterentwickelt.

Weitere XML-Anfragesprachen

Neben den oben beschriebenen weit verbreiteten XML-Sprachen gibt es noch einige andere Projekte und Sprachen, die interessante weitere Aspekte aufzeigen.

Die erste XML-Anfragesprache, die selber XML-Syntax verwendete, wurde in [Moer00] vorgestellt: Ähnlich wie XSLT definiert *YAXQL* Sprachelemente, die XPath-Ausdrücke als Attributwerte und Elementinhalt einbetten. Das Ergebnis wird ebenfalls durch YAXQL-Befehlselemente erzeugt. Wie auch für XML-QL und XQuery werden im Selektionsteil Variablenbindungen erzeugt, die an den Konstruktionsteil übergeben werden.

XML-GL [CCDF99] ist eine grafische XML-Anfragesprache nach dem »Query by Example"-Prinzip. XML-GL verwendet das *XML Graphical Data Model* (XML-GDM), in dem sowohl DTDs als auch XML-Instanzen repräsentiert werden. Die Sprache basiert direkt auf der grafischen Darstellung und verwendet *extract-match-clip-construct*-Anfragen. Die Aufgaben der einzelnen Schritte sind denen in XML-QL und XQuery sehr ähnlich.

XPathLog [May01] ist eine direkt XPath-basierte XML-Anfrage-, Datenmanipulations- und -integrationssprache im Datalog-Stil. Das Design vereinigt die Konzepte von F-Logic und XPath. Dabei wird XPath um Variablenbindungen erweitert, mit denen Knoten an jeder Stelle des Pfadausdrucks selektiert und an Variablen gebunden werden können. Weiterhin wird eine konstruktive Semantik für XPath-Ausdrücke in Regelköpfen definiert, um Updates auszudrücken. Mit XPathLog lassen sich auch rekursive Anfragen formulieren. XPathLog ist in dem LoPiX-System implementiert.

XML-basierte Systeme

MIX (*Mediation in XML*) [BGLM99] verwendet die Sprache Xmas (*XML Matching und Structuring*), die wiederum von XML-QL abgeleitet ist, zur Integration von XML-Daten. Zusätzlich zu Xmas als Anfragesprache wird ein grafisches Benutzerinterface bereitgestellt.

Das Produkt Tamino [Tamino] der Software AG ist eine kommerzielle XML-Plattform für Electronic Business, bestehend aus Speicherungs-, Entwicklungs- und Integrationskomponenten für XML-Daten und Anwendungen. Die Anfrageschnittstelle verwendete zuerst XQL und wurde später an den XPath-Standard angepasst. Es werden Werkzeuge zur Erzeugung und Veränderung von XML-Dokumenten angeboten.

eXcelon [Excelon] ist ein weiteres XML-basiertes B2B-Produkt. Es bietet eine Anfrageschnittstelle für XPath (in den ersten Versionen noch für XQL) und für XSLT-Stylesheets. Eine beachtenswerte Erweiterung von eXcelon ist XUL (*XML Update Language*), womit XSLT um Manipulationskonstrukte erweitert wird: »Updategrams« sind XSLT-Instanzen, die spezielle Elemente enthalten, deren

Inhalt aus <foreach>, <update> und <remove>-Elementen, die Veränderungen an der Datenbank spezifizieren, besteht.

Die Unterstützung von XML in Datenbanksystemen wird in Kapitel 13 detailliert beschrieben.

XML in relationalen Datenbanken

Die Hersteller relationaler Datenbanken wählen zwei Wege, um in XML abgefasste Dokumente in einer relationalen Datenbank abzuspeichern und zu verarbeiten. Der konzeptuell einfachste Weg ist die Ablegung des gesamten Dokumentes als ein Attributwert in Form eines CLOB (*Character Large Object*). Der zweite Weg besteht aus der Analyse des Dokumentes und anschließender Speicherung in einer oder mehreren Relationen. Nachteil der ersten Variante ist, dass zum Verarbeiten des Dokumentes die Erweiterung von SQL um zusätzliche Funktionalität zum Auffinden der Elemente, Attribute und Tags innerhalb des CLOBs sowie zur weiteren Auflösung der inneren Struktur der Attributwerte notwendig ist. Die zweite Variante erlaubt die Verarbeitung der relationalen Darstellung des Dokumentes mittels SQL. Nachteilig ist hier jedoch der durch die Abbildungsvorschrift entstehende zusätzliche Aufwand.

Die Abbildung zwischen dem relationalen Datenmodell und XML wurde in mehreren Projekten untersucht. *SilkRoute* [FeTS00] ist eng mit XML-QL verwandt. Das Hauptinteresse liegt hier in der automatischen Konvertierung von relationalen Daten nach XML zu einer vorgegebenen DTD. Dabei wird XML-QL zu RXL (*Relational to XML Transformation Language*) erweitert. Auf der relationalen Seite entspricht RXL SQL, und auf der XML-Seite XML-QL. In *Xperanto* [CFIL00] wird eine zugrunde liegende Datenbank auf eine XML-Sicht abgebildet, auf deren Basis weitere Benutzersichten definiert werden. Diese werden intern in SQL-Anfragen übersetzt. Xperanto soll das IBM-DB2-Datenbanksystem um XML-Funktionalität ergänzen.

Nähere Informationen zum Verhältnis XML und relationale Datenbanken finden sich in Kapitel 5 oder auch in [Ende01].

Eine direkte Verbindung zwischen SQL und XML wird durch das SQL-Standardisierungskomitee der ISO in SQLX [EiMe01,SQLX02] angestrebt. Hierbei wird »XML« als zusätzlicher Datentyp definiert und eine Syntax zur Erzeugung von Strukturen dieses Typs, d.h. XML-Fragmenten angegeben (basierend auf den XQuery-Konstruktoren). Weiterhin werden Abbildungen zwischen XML/XML-Schema-Datentypen und SQL-Datentypen sowie auf Instanzebene definiert (d.h. zwischen Relationen und Tupeln einerseits und XML-Fragmenten andererseits). Eine direkte Einbindung von XQuery-Anfragen wird bisher nicht angegeben.

3.9 Zusammenfassung und Ausblick

In diesem Kapitel wurde die Entwicklung von XML-Anfragesprachen sowie verwandter Funktionalität beschrieben. XPath (siehe Abschnitt 3.2) hat sich als

Navigationsformalismus etabliert: Version 1.0 wurde 1999 als Empfehlung des W3C verabschiedet [XPath99] und dient seitdem als Basis für viele weitere Konzepte im XML-Bereich und wird von allen gängigen XML-Systemen unterstützt (siehe Kapitel 13). Version 2.0 bietet weitere Ergänzungen [XPath01]. Die auf XPath basierende funktionale, regelbasierte Transformationssprache XSLT [XSLT01] (siehe Abschnitt 3.7) ist ebenfalls schon etabliert. Im Gegensatz dazu befindet sich die auf XPath aufbauende Anfragesprache *XQuery* (siehe Abschnitt 3.5) noch im Status eines W3C Working Draft; die Verabschiedung von XQuery 1.0 als Empfehlung ist absehbar. In diesem Zeitraum werden auch Implementierungen verfügbar sein. XQuery 1.0 wird dabei eine reine Anfragesprache bleiben. Die – auch als XUpdate bezeichnete – Spracherweiterung, um Veränderungen an XML-Daten durchführen zu können (wie in Abschnitt 3.6 beschrieben), wird später folgen. Diese Komplettierung der Funktionalität – wenn man die Situation etwa mit SQL vergleicht – schließt das grobe Sprachdesign ab. Weiterhin bleiben dann verschiedene Detailaufgaben zu lösen, um das Gesamtbild zu vervollständigen. Einige davon, z.B. *XQueryX* als XML-Syntax zu XQuery, oder das *Data Model* und die eng damit vorhandene *Formal Semantics* als Grundlage für theoretische Untersuchungen sowie implementierungstechnische Aspekte (Optimierung) sind schon weit fortgeschritten. Zu den weiteren Aspekten, die im Umfeld einer Anfragesprache zu suchen wären, gehören dann z.B. noch ein Transaktionskonzept sowie Zugriffskontrolle. Einige verwandte Themenbereiche werden in nachfolgenden Kapitel genauer untersucht: Speicherungsaspekte werden in Kapitel 5 beschrieben, insbesondere wird in Kapitel 8 auf Indexierungsmöglichkeiten eingegangen. Kapitel 13 beschreibt einige XML-Datenbanksysteme und Kapitel 14 geht auf das Benchmarking von XML-Anwendungen ein.

Literatur

[AQMW] S. Abiteboul, D. Quass, J. McHugh, J. Widom und J. Wiener. *The Lorel Query Language for Semistructured Data*. In Intl. Journal on Digital Libraries (JODL), 1(1), 1997.

[BGLM99] C. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov und V. Chu. *XML-Based Information Mediation with MIX*. In ACM Intl. Conference on Management of Data (SIGMOD), 1999.

[CCDF99] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi und L. Tanca. *XML-GL: A Graphical Language for Querying and Restructuring XML Documents*. In Proc. 8th International World Wide Web Conference (WWW 8), 1999.

[CDSS98] C. Delobel, J. Siméon und K. Smaga. *Your Mediators need Data Conversion*. In ACM Intl. Conference on Management of Data (SIGMOD), 1998.

[CFIL00] M. J. Carey, D. Florescu, Z. G. Ives, Y. Lu, J. Shanmugasundaram, E. Shekita und S. Subramanian. XPeranto: *Publishing Object-Relational Data as XML*. WebDB 2000, pp. 105-110, 2000. Siehe auch: http://www7b.software.ibm.com/dmdd/library/demos/0203xperanto/0203xperanto.html.

[ChCS00] V. Christophides, S. Cluet und J. Siméon. *On Wrapping Query Languages and Efficient XML Integration*. In ACM Intl. Conference on Management of Data (SIGMOD), pp. 141-152, 2000.

[ChRF00] J. Robie und D. Florescu. Quilt: *An XML Query Language for Heterogeneous Data Sources*. In WebDB 2000, pp. 53-62, 2000. Siehe auch http://www.almaden.ibm.com/cs/people/chamberlin/quilt.html.

[DFFL98a] A. Deutsch, M. Fernandez, D. Florescu, A. Levy und D. Suciu. *XML-QL: A Query Language for XML*. http://www.w3.org/TR/NOTE-xml-ql (1998) und 8th. WWW Conference, 1999.

[DFFL98b] A. Deutsch, M. Fernandez, D. Florescu, A. Levy und D. Suciu. XML-QL: *A Query Language for XML*. http://www.research.att.com/sw/tools/xmlql, 1998.

[DFFL99] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Maier, and D. Suciu. *Querying XML Data*. In IEEE DATA Engineering Bulletin, 22(3), 1999.

[EiMe01] A. Eisenberg and J. Melton. *SQL/XML and the SQLX Informal Group of Companies*. In SIGMOD Record, 30(3):105-108, 2001. Siehe auch www.sqlx.org.

[Ende01] Jost Enderle. *XML in relationalen Datenbanken*. In Informatik Spektrum 24(6), pp. 357-368, 2001.

[Excelon] eXcelon Corp. *XML Application Development using eXcelon*. http://www.exceloncorp.com, 2001.

[FeTS00] M. Fernandez, W.-C. Tan und D. Suciu. *Silk Route: Trading between Relations and XML*. In Proc. 9th International World Wide Web Conference (WWW 9), 2000.

[FFKL98] M. F. Fernandez, D. Florescu, J. Kang, A. Y. Levy und D. Suciu. *Catching the Boat with Strudel: Experiences with a Web-Site Management System*. In ACM Intl. Conference on Management of Data (SIGMOD), 1998.

[FFLS97] M. Fernandez, D. Florescu, A. Levy und D. Suciu. *A Query Language for a Web-Site Management System*. In SIGMOD Record, 26(3):4-11, 1997.

[GMPQ97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos und J. Widom. *The TSIMMIS Approach to Mediation: Data Models and Languages*. Journal of Intelligent Information Systems, 8(2), 1997.

[GoMW99] R. Goldman, J. McHugh und J. Widom. *From Semistructured Data to XML: Migrating the Lore Data Model and Query Language*. In WebDB 1999, 1999.

[HuMa99] G. Huck and I. Macherius. *GMD IPSI XQL Engine*. http://xml.darmstadt.gmd.de/xql, 1999.

[Kay00] M. Kay. *XSLT : Programmer's Reference*. Wrox Press, 2000.

[KiLa89] M. Kifer and G. Lausen. F-logic: *A Higher-Order Language for Reasoning About Objects, Inheritance and Scheme*. In Proc. ACM SIGMOD Intl. Conference on Management of Data, pp. 134-146, 1989.

[KiLW95] M. Kifer, G. Lausen und J. Wu. *Logical Foundations of Object-Oriented and Frame-Based Languages*. Journal of the ACM, 42(4):741-843, July 1995.

[Leht01] P. Lehti. *Design and Implementation of a Data Manipulation Processor for an XML Query Language*. Master's thesis, Technische Universität Darmstadt, 2001.

[MAGQ97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass und J. Widom. *Lore: A Database Management System for Semistructured Data*. SIGMOD Record, 26(3):54-66, 1997.

[Mondial] W. May. *The Mondial Database*, 2001. http://www.informatik.uni-freiburg.de/~may/Mondial.

[May01] W. May. *XPath-Logic and XPathLog: A Logic-Based Approach for Declarative XML Data Manipulation*. Habilitation Thesis, Universität Freiburg, 2001. Siehe auch http://www.informatik.uni-freiburg.de/~may/lopix.

[May02] W. May. *Querying Linked XML Document Networks in the Web*. In 11th. WWW Conference, 2002.

[MaBe01] W. May and E. Behrends. *On an XML Data Model for Data Integration*. In Intl. Workshop on Foundations of Models and Languages for Data and Objects (FMLDO 2001). Erscheint in Springer LNCS.

[Moer00] G. Moerkotte. *YAXQL: A Powerful and Web-Aware Query Language Supporting Query Reuse*.
http://pi3.informatik.uni-mannheim.de/~moer/myself.html, January 2000.

[RLS98] J. Robie, J. Lapp und D. Schach. *XML Query Language (XQL)*. In QL'98 – The Query Languages Workshop; http://www.w3.org/TandS/QL/QL98/pp/xql.html, 1998.

[Rob99] J. Robie. *XQL (XML Query Language)*.
http://www.metalab.unc.edu/xql/xql-proposal.html, 1999.

[Quip01] Software AG. *QuiP: An XQuery Implementation*.
http://www.softwareag.com/developer/quip, 2001.

[Tamino] Software AG. *Tamino – An Internet Database System*.
http://www.tamino.com, 2001.

[TIHW01] I. Tatarinov, Z. G. Ives, A. Halevy und D. Weld. *Updating XML*. In ACM Intl. Conference on Management of Data (SIGMOD), 2001.

[Wadl99a] P. Wadler. *A Formal Semantics of Patterns in XSLT*. In Markup Technologies, 1999. http://www.cs.bell-labs.com/who/wadler/topics/xml.html.

[Wadl99b]P. Wadler. *Two Semantics for XPath*. 1999.
http://www.cs.bell-labs.com/who/ wadler/topics/xml.html.

[XMLInf01] *XML Information Set*. http://www.w3.org/TR/XML-infoset, 1999-2001.

[XMLQR01] *XML Query Requirements*. http://www.w3.org/TR/xmlquery-req, 2001.

[XPath99] *XML Path Language (XPath) Version 1.0*.
http://www.w3.org/TR/xpath, 1999.

[XPath01] *XML Path Language (XPath) Version 2.0*.
http://www.w3.org/TR/xpath, 2001.

[XPtr01] *XML Pointer Language (XPointer)*. http://www.w3.org/TR/xptr, 1999-2001.

[XQFS01] *XQuery 1.0 Formal Semantics*. http://www.w3.org/TR/query-semantics, 2001.

[XQPDM01] *XQuery 1.0 and XPath 2.0 Data Model*.
http://www.w3.org/TR/query-datamodel, 2001.

[XQPFO01] *XQuery 1.0 and XPath 2.0 Functions and Operators*.
http://www.w3.org/ TR/xquery-operators, 2001.

[XQuery01] *XQuery: A Query Language for XML*.
http://www.w3.org/TR/xquery, 2001.

[XQX01] *XML Syntax for XQuery 1.0 (XQueryX)*.
http://www.w3.org/TR/xqueryx, 2001.

[XSL01] *Extensible Stylesheet Language (XSL)*.
http://www.w3.org/Style/XSL/, 1998-2001.

[XSLT01] *XSL Transformations (XSLT)*. http://www.w3.org/TR/xslt, 1999-2001.

Teil II: Architektur und Implementierung

4 Architektur von Web-Informationssystemen

Gerti Kappel, Werner Retschitzegger, Birgit Pröll, Rainer Unland, Bahram Vojdani

Kurzfassung

Web-Informationssysteme (WebIS) haben sich in letzter Zeit immer mehr in Richtung vollwertiger Softwareanwendungen entwickelt, die interaktive, datenintensive und individualisierbare Dienste über verschiedene Endgeräte zur Verfügung stellen. Dieses Kapitel gibt einen Überblick über verschiedene Architekturen für WebIS, die diesen unterschiedlichen Entwicklungen Rechnung tragen. Für die Realisierung dieser Architekturen kann eine Vielzahl verschiedener Basistechnologien eingesetzt werden, wobei der Fokus zunächst auf Alternativen zur Datenbankanbindung gelegt wird. Im Anschluss daran wird eine an der Entwicklungshistorie von WebIS orientierte Klassifikation von WebIS-Architekturen vorgenommen, wobei die zuvor behandelten Technologien entsprechend zugeordnet werden.

4.1 Einführung und Begriffsdefinitionen

Das Internet stellt ein verteiltes Informationssystem dar. Eine Vielzahl von Rechnern sind in Client/Server-Beziehungen über Kommunikationskanäle und -protokolle miteinander verknüpft. Die vielen Dienste im Internet, wie z.B. das WWW, sind durch eine Anzahl hierarchisch angeordneter Protokolle (bzw. Protokollebenen) umgesetzt. Man spricht von einer *Protokollhierarchie*. Eine höher angesiedelte Ebene bietet – von den Eigenschaften der niedrigeren Ebenen abstrahierend – komplexere Dienste an. Das *Hyper Text Transfer Protocol* (HTTP) [FGMF97]) ist bereits ein sehr hochwertiger Dienst. Er hat die Aufgabe, ein *gemeinsam verständliches Schema* für die Datenübertragung zu definieren. Dadurch kann der eine Kommunikationspartner die Informationen/Daten des anderen auswerten. Daten und Informationen werden dabei in einem so genannten *multimedialen Hypertext-Dokument* präsentiert. Dieses wird mit Hilfe der *Hyper Text Markup Language* HTML [Ragg96]) beschrieben. Auf dem Rechner des Web-Client installierte Browser lokalisieren solche auf so genannten Web-Servern abgelegte Dokumente über einen Verweis (Uniform Resource Locator, URL) der Form *Dienst://Server/Verzeichnis/Datei* und stellen sie am Bildschirm dar (siehe [BeGr98]). Über im Dokument enthaltene Verweise kann (rekursiv) auf andere Dokumente oder Ressourcen zugegriffen werden. Da in der URL auch andere

Internetdienste verwendet werden können, wurde so eine einheitliche Schnittstelle zum Internet geschaffen.

Wie bereits gesagt erfolgt die Kommunikation zwischen Web-Client und Web-Server über das HTTP-Protokoll. Der Server liest die angeforderte Information aus seinem Dateisystem und überträgt sie zum Client. Die Daten enthalten eine Formatbeschreibung in Form von *MIME* (Multipurpose Internet Mail Extensions; [Seeg96]). Auf dieser Basis entscheidet der Browser, ob er die gesendete Bytefolge selbst interpretiert (z.B. MIME-Typ text/html) oder andere Hilfswerkzeuge (Helpers, z.B. application/excel) und Zusätze (Plug-ins, z.B. video/quicktime) zur Interpretation und Darstellung auf dem Bildschirm aufruft. Diese statische Variante, bei der lediglich vorkonfigurierte Webseiten übertragen werden, muss bei anspruchsvolleren Anwendungen durch eine Möglichkeit ergänzt werden, Seiten dynamisch zu konfigurieren. Heutige offene und leistungsfähige Web-Browser erlauben die integrierte Darstellung verschiedener Medientypen, bei denen beispielsweise die Ergebnisse von Datenbankanfragen zusammen mit statischen Texten, Bildern und Videosequenzen dynamisch generiert und dargestellt werden können.

Der Weg von *statischen Dateien* zur *dynamischen Dokumentdarstellung* wird durch die Integration von Programmen bzw. Programmaufrufen in Webseiten geebnet. Dies erlaubt einerseits das dynamische Generieren von Webseiten und andererseits auch die Integration bereits bestehender Datenquellen (über die Ausführung des möglicherweise in einer beliebigen Programmiersprache geschriebenen Programms). Für den Client-Browser stellt sich das dynamisch generierte Dokument wie ein statisches Dokument dar. Der Weg der Integration von in Datenbanksystemen gehaltenen (Legacy-)Daten in Hypertext-Dokumente ist damit aufgezeigt.

Zum Verständnis der weiteren Diskussion ist es notwendig, die Begriffe zustandslose/zustandswahrende Verbindung zu verstehen. Eine *zustandslose Verbindung (Zustandslosigkeit)* ist gegeben, falls nach dem Übermitteln einer Datei bzw. von Daten an den WWW-Client die Verbindung zum (Datenbank-)Server wieder abgebaut wird. Somit kann es keine HTTP-Sitzung geben, bei der sich der Benutzer lang andauernd auf einem WWW-Server (einer Datenbank) einloggt, um beispielsweise durch das dort befindliche Informationsangebot zu surfen. Vielmehr wird für jede einzelne Datenbankanfrage eine separate Verbindung aufgebaut und nach der Datenübertragung wieder geschlossen. Die Vorteile der Zustandslosigkeit liegen auf der Hand. Der Server muss sich keine Details über die Verbindungen zu den Clients merken. Somit wird sowohl Speicher als auch Rechenzeit eingespart. Das ist insbesondere bei Servern mit sehr hohem Datenaufkommen wichtig (wie z.B. bei Suchmaschinen). Denn hier müssen sehr schnell sehr viele Anfragen bearbeitet werden.

Andererseits fehlt damit auch die Möglichkeit, sich Informationen über den Client zu merken, um z.B. besser auf dessen Anfragen reagieren zu können. Dies kann insbesondere Probleme bringen, wenn zunächst eine Assoziativsuche auf der DB durchgeführt wird und in einer darauf folgenden Datenbankoperation in

Abhängigkeit von dieser Anfrage Änderungen durchgeführt werden. Da jede nachfolgende Anfrage und damit auch das Datenbanksystem (DBS) nichts mehr von dem vorherigen Zugriff weiß, kann es beispielsweise auch keine Transaktion geben, die die gesamte Folge von (zusammengehörigen) Operationen schützt. Dies kann zu Konsistenzproblemen führen. Auch kann insbesondere die Datenmanipulation zu Performanzproblemen führen, da die Datenintegrität und Konsistenz der DB gesichert sein müssen. Die dazu notwendigen Exklusivzugriffe auf Daten können zeitintensiv sein, so dass es zu gewissen Wartezeiten kommen kann, bis das DBS die erfolgreiche Ausführung der Änderungen an die Anwendung gemeldet hat. Im Folgenden wird deshalb insbesondere auf Techniken eingegangen, die es erlauben, Operationen auf dem DBS zu bündeln, d.h. eine *zustandswahrende Verbindung* aufzubauen.

In den letzten Jahren wurden zahlreiche alternative Techniken zur Client/Web(DBS)-Server-Anbindung entwickelt ([Aper94], [BeGr98], [EhKR98], [Löse98], [MaKH00], [NgSr96], [Shkl96], [Tura99]), deren wichtigste Vertreter in Abb. 4-1 im Überblick dargestellt sind.

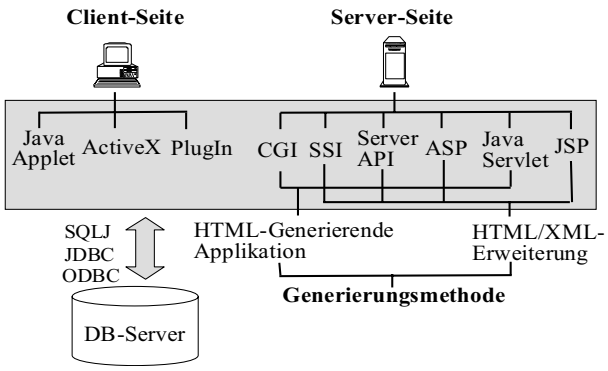


Abb. 4-1: *Techniken zur Client/Web(DBS)-Server-Anbindung*

Abhängig von den Anforderungen des konkreten Anwendungsbereichs ist auch eine Kombination verschiedener Techniken innerhalb eines WebIS möglich. Die Techniken als solche können grundsätzlich in *clientseitige* und s*erverseitige* unterteilt werden. Im ersten Fall wird die Applikationslogik, die auch die Datenbankzugriffe realisiert, zum Client transferiert und dort ausgeführt. Dies entspricht einer Übertragung des Codes. Im zweiten Fall wird die Anwendungslogik am Server ausgeführt, was bedeutet, dass die (fertigen) Ergebnisse übertragen werden.

Dieses Kapitel gibt einen Überblick über verschiedene Architekturen für WebIS, die diesen unterschiedlichen Entwicklungen Rechnung tragen. In Abschnitt 4.2 werden zunächst die Anforderungen an eine Web/Datenbanksystem-Anbindung diskutiert. Da Datenbanksysteme eine Schlüsselrolle in WebIS einnehmen, werden in den Abschnitten 4.3, 4.4 und 4.5 alternative Technologien zur Web/DBS-Anbindung vorgestellt. Es wird zwischen *client-* und *serverseitigen*

und *übergreifenden* Technologien unterschieden. Abschnitt 4.6 liefert eine Klassifikation der Architekturen von WebIS, wobei insbesondere auch die spezifische Eignung jeder Architekturvariante für spezifische Anwendungen anhand verschiedener Kriterien diskutiert wird. Eine Zusammenfassung schließt das Kapitel ab.

4.2 Anforderungen an eine Client/Web(DBS)-Server-Anbindung

Das Internet ist zwischenzeitlich zu einem Pool für verschiedenartigste Dienste geworden. Einerseits bietet es einzelnen Personen, beispielsweise einem Forscher, die Möglichkeit, einen begrenzten Service der Allgemeinheit zur Verfügung zu stellen. Andererseits ist das Internet aber auch die grundlegende Präsentationsplattform von großen internationalen Unternehmen (wie Amazon, ebay) geworden. Aus dieser hohen Heterogenität ergeben sich auch völlig divergierende Anforderungen. Während beispielsweise auf Forschungsergebnisse vielleicht täglich einmal zugegriffen wird, liegt das Transaktionsvolumen bei großen E-Commerce-Unternehmen eher bei Tausenden von Transaktionen pro Sekunde. Damit hängt deren Überleben substanziell von einer mit steigenden Kundenbedürfnissen/-aufkommen unmerklich wachsenden Technologieplattform ab. Kunden, die am Bildschirm lange auf Resultate warten müssen, kommen nicht wieder. *Performanz* ist aber nur eine Seite. Die andere Seite ist die *Dynamik*. Unser Forscher kann als Service seine Forschungsarbeiten anbieten. Sie sind wohl eher als statische Dokumente abgelegt, die dann auch in dieser Form heruntergeladen werden. Ein E-Commerce-Unternehmen lebt aber von der hohen Individualisierbarkeit seiner Dienste. Jeder Kunde will die Informationen exakt so sehen, wie es ihm gefällt. Dabei ist auch noch ein hoher Sicherheitsstandard zu garantieren, beispielsweise bei der Übertragung von Kontoinformationen. Dies sind nur einige der divergierenden Anforderungen, die aus den verschiedensten Web-Diensten resultieren. Im Folgenden sollen mögliche weitere Anforderungen an eine Client/Web(DBS)-Server-Anbindung kurz diskutiert werden.

Integrierte Benutzerschnittstelle

Eine offene und leistungsfähige Benutzerschnittstelle erlaubt die integrierte Darstellung verschiedener Medientypen. Beispielsweise können die Ergebnisse von Datenbankanfragen zusammen mit statischen Texten, Bildern und Videosequenzen durch einen modernen Browser dargestellt werden. Voraussetzung ist aber, dass die Daten/Informationen in einer aus Client-Sicht interpretier- und darstellbaren Form geliefert werden. Proprietäre Formate werden immer einen bestimmten Anteil des potenziellen Benutzerkreises ausschließen.

Interaktivität

Insbesondere bei komplexeren Anwendungen kann der Wunsch des Benutzers bestehen, interaktiv Änderungen vorzunehmen. Beispielsweise könnte bei einer Anfrage der Suchraum weiter eingeschränkt werden. Interaktivität verlangt fast

zwangsläufig eine zustandswahrende Verbindung, soll sie performant und konsistent unterstützt werden.

Konsistenz und Datenintegrität

Bei einer Datenmanipulation oder bei wiederholten Anfragen an die DB muss die Datenkonsistenz gesichert sein. Dies stellt durchaus ein schwerwiegendes Problem dar, das vor allem bei verteilten Informationssystemen auftritt, wie nicht zuletzt die umfangreichen Diskussionen diesbezüglich im Bereich der verteilten DBS zeigen. Die Zustandslosigkeit von HTTP und die Tatsache, dass der Browser und das bzw. die DBS sehr unabhängig voneinander operieren, erschwert dessen effiziente Nutzung als Frontend erheblich.

Performanz

Die zu erwartende Performanz von den verschiedenen Lösungen kann sehr unterschiedlich sein. Ein Vergleich ist aufgrund der Unterschiede in den Datenbank- bzw. Anbindungsarchitekturen sehr schwierig und hängt nicht zuletzt auch von der konkret gegebenen Anwendungssituation ab. Hier helfen nur spezifisch zugeschnittene Leistungsmessungen (Benchmarks).

Sicherheit

Gerade der Bereich des E-Commerce hängt ganz entscheidend von der Frage ab, wie die Sicherheit, d.h. Vertraulichkeit der Informationen sichergestellt werden kann, die über das Internet fließen. Aber nicht nur hier sind Bedenken von Seiten des Benutzers zu erwarten. Manche Technologien (vor allem die von Microsoft unterstützten) erlauben den von einem Web-Server geladenen Programmen lokal beliebige Operationen auszuführen. Damit ist ein Ausspähen des Benutzers und damit der gläserne Mensch keine Utopie mehr. Auch der Diebstahl von intellektuellem Eigentum ist damit möglich. Die Sicherheitsproblematik ist ein sehr heikles Thema, da sie einerseits die Privatsphäre des Benutzers bedroht, andererseits die gebotenen Möglichkeiten zum Schutz derselben die Geduld des Benutzers oft überfordern, so dass sich dieser eher dadurch abstoßen lässt. Grundsätzlich gilt, dass das Sicherheitskonzept Java-basierter Lösungen besser ist als das von Microsoft-basierten Lösungen, da importierte Java-Programme nur sehr eingeschränkten Zugriff auf lokale Ressourcen haben, während dies bei Microsoft-basierten Lösungen i.d.R. nicht der Fall ist.

Skalierbarkeit

Eine Client/Web(DBS)-Server-Anbindung sowie ein evtl. dahinter stehender Zugriff auf ein oder mehrere DBS muss sich auch bei steigenden Anforderungen als leistungsfähig erweisen, d.h. sie/er muss bei steigenden Anforderungen in ihrer/seiner Leistungsfähigkeit anpassbar sein. Die Anzahl von gleichzeitig in akzeptabler Zeit ausführbaren Zugriffen und die Größe der Datenbank sind zwei wichtige Aspekte für die Skalierbarkeit eines Systems.

Offenheit

Die Offenheit wird in Bezug auf die Einfachheit der Integration verschiedener Datenbanken, verschiedener Datenmanipulationssprachen und neuer Browser gemessen. Offenheit und Performanz sind dabei leider i.d.R. sich widersprechende Eigenschaften. Eine gute Performanz wird vor allem durch eine sauber auf die spezifische Umgebung zugeschnittene Lösung erzielt, wobei die Kopplung zwischen Diensten und der Anwendungsumgebung möglichst eng sein soll. Dies widerspricht aber der Offenheit, bei der man eine Lösung oft auf einer sehr abstrakten Ebene anbietet, die dann auf die konkrete Ebene abgebildet werden muss. Diese Abbildung kostet nicht nur Performanz, sondern verhindert auch, dass die spezifischen Stärken einer spezifischen Umgebung (z.B. eines Betriebssystems) ausgenutzt werden können.

Ein gutes Beispiel für diese Problematik sind hier die Java-basierten und Microsoft-zentrierten Lösungen. Erstere sind weitestgehend betriebssystem- und plattformunabhängig, dafür aber eben auch nicht so effizient (da der Code letztendlich immer interpretiert werden muss). Letztere sind deutlich performanter, dafür aber nur eingeschränkt nutzbar (eben nur auf Intel-Plattformen unter einem Microsoft-Betriebssystem). Diese Diskrepanz wird sich auch bei der folgenden Diskussion der verschiedenen Techniken immer wieder zeigen.

4.3 Clientseitige Technologien

Bei der *clientseitigen Anbindung* wird der Code, der den Datenbankzugriff und Teile des WebIS realisiert, zum Client transferiert und dort ausgeführt. Dazu wird eine direkte Verbindung zwischen dem Client und der Datenbank in der Serverumgebung aufgebaut. Die Funktionsfähigkeit und Performanz des WebIS ist hierbei von der hardware- und softwaremäßigen Ausstattung des Clients abhängig, z.B. von der Installation eines für die Applikation benötigten Plug-ins in der richtigen Version, und liegt somit zum Teil außerhalb des Einflussbereiches der DB-Server. Außerdem kann die Ladezeit und die Netzbelastung bei größeren WebIS beträchtlich sein. Diese Anbindungsalternative bietet andererseits einige Vorteile, wie erweiterte Möglichkeiten der Oberflächenfunktionalität auf der Präsentationsebene durch proprietäre GUI-Elemente anstatt HTML, sowie die Bereitstellung von Zustandsinformation über mehrere HTTP-Anfragen hinweg durch Verwendung proprietärer Datenbank-Protokolle anstelle des zustandslosen HTTP-Protokolls [HTTP01]. Der Einsatz clientseitiger Datenbankanbindungen erscheint daher vorwiegend für Intranet-Anwendungen sinnvoll. Java-Applets sind der am weitesten verbreitete Ansatz der clientseitigen Datenbankanbindung. Außerdem zählen neben ActiveX auch spezielle Client-Plug-ins und externe Viewer dazu, bei denen das entsprechende Programm am Client installiert sein muss.

4.3.1 Java-Applets

Java ist eine plattformunabhängige Sprache, die von einer virtuellen Maschine (Java Virtual Machine, JVM) interpretiert wird. Der Name *Applet* steht für »kleine Anwendung«. Java-Applets sind am Web-Server abgelegte Java-Programme, die bei Aufruf dynamisch – in eine HTML-Seite eingebettet – zum Web-Client transferiert und dort durch die JVM ausgeführt werden (siehe Abb. 4-2). Damit ist z.B. die Integration einer Animation auf der Client-Seite ebenso möglich wie der Zugriff auf Spreadsheets oder Datenbanken.

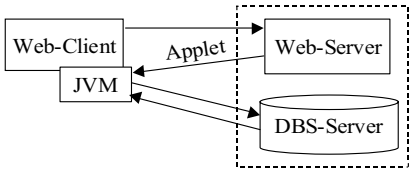


Abb. 4-2: *Web/DB-Anbindung über Java-Applets*

Bei den Java-Entwicklungssystemen wird eine JVM mitgeliefert, die bei Bedarf über das Java-Plug-in geladen werden kann. Besser wäre es, wenn die JVM bereits überall dort lokal vorinstalliert wurde, wo Applets ausgeführt werden sollen. In modernen HTML-Browsern ist die JVM direkt integriert.

Applets können vom Client aus auf beliebige Datenbanken zugreifen (siehe Abb. 4-2). Dies geschieht üblicherweise über JDBC oder SQLJ (siehe Kapitel 4.5). Die Verbindung kommt dabei nicht über HTTP zustande, sondern über ein eigenes Protokoll (wie JDBC API). Die Zugriffsmöglichkeiten und die Funktionalität des Java-Applets sind hierbei aus Sicherheitsgründen eingeschränkt bzw. werden in einer Sicherheitspolitik definiert und durch eine Instanz der Klasse *SecurityManager* überwacht.

Zum Beispiel muss die Datenbank auf demselben Web-Server wie das Java-Applet liegen. Diese Einschränkungen wurden jedoch durch das Konzept der »trusted« Applets etwas relativiert. Der Sprachumfang von Java sowie zahlreiche existierende Java-Bibliotheken, die z.B. interaktive GUIs effizient und komfortabel realisieren lassen, ermöglichen eine rasche Umsetzung einfacher WebIS.

Diese Lösung hat eine Reihe von Vorteilen. Da die Datenbankverbindung länger aufrechterhalten werden kann, steht der Realisierung von aus mehreren DB-Aktionen zusammengesetzten Transaktionen nichts mehr im Weg. Der Browser kann nun als Ein-/Ausgabeinstrument einer komplexen Anwendung genutzt werden. Integritätsbedingungen können schon direkt bei der Eingabe auf der Client-Seite überprüft werden und nicht erst bei der Anfrageausführung. Auch sind die entwickelten Programme plattformunabhängig [Löse98]. Schließlich werden alle Datenbankaufrufe ohne den Umweg über den Web-Server direkt an den Datenbank-Server geschickt, was wegen der direkten Kommunikation deutliche Performanzvorteile mit sich bringt [Löse98].

Dem stehen aber auch einige Nachteile gegenüber. Das Applet kann keine Eingaben aus HTML-Formularen übernehmen oder Resultate in solche einfügen, was zu einer eingeschränkten Web-Integration führt. Jedoch können mit Hilfe von JavaScript-Anweisungen ([JaSc99]) Daten an das Applet übergeben werden. Dies ist allerdings – wie auch der umgekehrte Weg vom Applet zu HTML-Dokumenten – nur *plattformabhängig* möglich. Netscape unterstützt beispielsweise eine eigene Klasse *JSobject*, die das Abfragen und Manipulieren von JavaScript und damit von HTML erlaubt.

Das Applet und alle von ihm benötigten Java-Klassen müssen beim erstmaligen Öffnen der HTML-Seite zunächst vom Web-Server geladen werden, was zu *längeren Ladezeiten* führen kann.

Ein weiterer Nachteil ist die *Sicherheitsproblematik*: Arbeitet man nicht in einem abgeschirmten Intranet, besteht bei der Übertragung ohne Verschlüsselung die Gefahr der Manipulation [Löse98]. Die Sicherheit von Applets basiert in JDK 1.0 und JDK 1.1 vor allem auf dem so genannten »Sandbox Modell". Ab JDK 1.1 sind auch *zertifizierte Applets* möglich (s.u.). Wegen der strengen Sicherheitsrichtlinien von Java können Applets normalerweise nur Verbindungen zu dem Server einrichten, von dem das Applet selbst geladen wurde. Das bedeutet, dass Datenbank-Server und Web-Server auf demselben System laufen müssen, was wegen der doppelten »Server-Belastung« zu Leistungsproblemen führen kann.

Zertifizierte Applets sind Applets, die in Archiven zusammengefasst werden. Diese erhalten ein Zertifikat (z.B. von Verisign, www.verisign.com, von XCert, www.x509.com, direkt vom Programmierer etc.). Der Benutzer benötigt davon eine Kopie und trägt diese als »legal« in seine »Datenbank« ein. Somit haben alle Applets, die dieses Zertifikat tragen, eine individuell festlegbare Form des lokalen Zugriffs. Zusätzlich benötigt der Empfänger ein Zertifikat des Senders (»susan.cer«) und eine Datenbank mit Verweisen auf die jeweiligen Schlüssel und Sicherheitspolitiken (»sigistore«).

```
keystore "sigistore";
grant signedBy "susan" {
   permission java.util.PropertyPermission "user.home", "read";
   permission java.io.FilePermission "${user.home}/newfile", "write";
};
```

Abb. 4-3: *Beispiel einer (lokalen) Sicherheitspolitik für ein Applet*

Im obigen Beispiel stellt *sigistore* den Namen für die Datenbank dar, in der die Rechte abgespeichert werden. *Susan* ist die Programmiererin des Applet und *permission* gibt die Rechte an, die das Applet auf dem »lokalen Rechner« hat.

4.3.2 ActiveX

ActiveX ([Chap96]) kann als eine Menge von lose gekoppelten Technologien angesehen werden, die es erlauben, Multimedia-Inhalte (Video, Audio usw.) in

Webseiten einzubinden. Es ist eine integrierte Weiterentwicklung von Microsofts *OLE (Object Linking and Embedding)* und *COM (Component Object Model)*. Neu hinzugekommen sind insbesondere Fähigkeiten der verteilten Ausführung, wie sie von DCOM (Distributed COM) ermöglicht werden. ActiveX wird auch gerne als Microsofts Version der Java-Applets angesehen. Im Gegensatz zu Letzteren ist ActiveX aber keine Programmiersprache, sondern kann eher als ein »Kochbuch« angesehen werden, welches Auskunft darüber gibt, wie Anwendungen Daten gemeinsam nutzen können bzw. wie Multimedia-Inhalte in Webseiten eingebunden und zur Verfügung gestellt werden können. Das so genannte *ActiveX-Kontrollobjekt* (controller) entspricht im Wesentlichen einem Java-Applet, kann aber im Gegensatz zu diesem in unterschiedlichen Programmiersprachen geschrieben sein (z.B. C, C++, Visual Basic oder Java). Eng verbunden mit ActiveX ist Microsofts Skriptsprache *VBScript*. Sie erlaubt die Integration von Web-Anwendungen, interaktiven Elementen und HTML-Seiten. Sie ist also das Pendant zu JavaScript.

Java-Applets besitzen den Vorteil der Plattformunabhängigkeit. ActiveX ist momentan dagegen nur auf der *Windows-Plattform* verfügbar und ist auch eng mit diesem Betriebssystem gekoppelt, d.h. kann dieses voll nutzen. Damit entsteht ein erhebliches *Sicherheitsproblem*, welches Microsoft mit Hilfe eines Registrierungsservices zu lösen versucht. Dieser Service erlaubt es einem Browser, vor dem Laden das Kontrollobjekt zunächst zu prüfen bzw. zu authentifizieren. Ein Vorteil von ActiveX gegenüber Java ist die Geschwindigkeit, da es für die Microsoft-Plattform optimiert wurde.

Obwohl ActiveX (Java-Applets) hier unter der Rubrik clientseitige Technologie eingeführt wird, können solche Kontrollobjekte auch auf dem Server ausgeführt werden.

4.4 Serverseitige Technologien

Bei der *serverseitigen Anbindung* wird eine Webseite aus HTML-Tags und Inhalten der Datenbank zusammengebaut, bevor sie über das HTTP-Protokoll zum Client gesendet wird. Beispiele hierfür sind *CGI*, *ServerAPI* (z.B. LiveWire von Netscape oder ASP von Microsoft), *SSI*, *Java Servlets* und *JSP* [Löse98]. Serverseitige Anbindungsarten sind gekennzeichnet durch die Eigenschaften des HTTP-Protokolls und durch die eingeschränkten Möglichkeiten von HTML. Aus der Verwendung von HTML resultiert eine eingeschränkte Oberflächenfunktionalität. Die Zustandslosigkeit des HTTP-Protokolls bedeutet, dass die Verbindung zwischen Client und Server nur für die Dauer einer Anfrageabwicklung besteht und Zustandsinformation über aufeinander folgende Anfragen desselben Clients sowie Information über den Client selbst auf der Server-Seite nicht verfügbar sind. Session-Management zur Speicherung der Client-Daten muss durch Techniken wie URL Extension, Hidden Fields, Cookies oder proprietäre Konzepte auf oft umständliche Art realisiert werden [Löse98].

HTTP Cookies

Cookies sind kleine Textdateien, welche vom Web-Server zum Browser gesendet und dort gespeichert werden [Cook02]. Der Web-Server kann auf die im Cookie abgelegten Informationen zugreifen. Durch Cookies können unterschiedliche Anfragen eindeutig einem Client zugeordnet werden. Dadurch kann der Server eine aktuell offene Datenbanksitzung weiterführen, anstatt immer wieder eine neue starten zu müssen. Ein anderes sinnvolles Einsatzgebiet ist das Einkaufen per WWW: In den Cookies wird dann der aktuelle Inhalt des Warenkorbs gespeichert.

URL-Kodierung

Eine weitere Möglichkeit besteht darin, eine Session durch das Mitführen einer Kennung in der URL zu identifizieren. Somit kann der Server mehrere Anfragen als zusammenhängend erkennen.

Hidden Fields

Bei dieser Technik existiert die Möglichkeit, die Zustandsinformationen in versteckten Formularfeldern in der URL zwischenzuspeichern.

Im Hinblick auf die Generierung der HTML-Seiten auf Basis des Datenbankinhalts verfolgen serverseitige Anbindungstechnologien zwei unterschiedliche Ansätze (siehe Abb. 4.1):

- *HTML-Erweiterung.* In HTML-Seiten werden die Funktionen zur Selektion des notwendigen Datenbankinhalts direkt eingebettet und vom Web-Server im Zuge eines Parsing-Vorganges aufgerufen. Dieser Ansatz entspricht der Idee der ursprünglich in den NCSA-Web-Server eingebauten *Server Side Includes* (SSI), HTML um einfache Funktionalität zu erweitern. SSI sind um spezielle Steuerungsbefehle in Syntax von HTML-Kommentaren angereicherte HTML-Seiten, die vom Web-Server bei einem Zugriff dynamisch ausgewertet werden. Auf diese Weise ist es möglich, einfache Programmfunktionalität, wie z.B. Betriebssystemfunktionen, aber auch CGI-Programme inklusive Datenbankzugriffe, in eine HTML-Seite einzubauen.
- *HTML-generierende Applikation.* Die HTML-Seiten werden von einer eigenen Applikation erzeugt, die sowohl die notwendigen HTML-Tags generiert, als auch Datenbankzugriffe durchführt und die HTML-Seite mit dem aus der Datenbank selektierten Inhalt füllt.

4.4.1 Common Gateway Interface (CGI)

Bei CGI-Skripten [CGI01] handelt es sich um ausführbare Programme, die in einer beliebigen Programmiersprache geschrieben sein können und selbstständig (auch ohne Web-Server) ablauffähig sind. Typischerweise werden Skriptsprachen wie Perl oder Tcl, aber auch C oder C++ verwendet.

Der Server erkennt anhand des Pfades der angefragten URL, dass ein CGI-Programm gestartet werden soll. Dieses realisiert den Datenbankzugriff (siehe Abb. 4-4), gibt die vom Benutzer eingegebenen Parameter weiter und liefert das Ergebnis – ein HTML Dokument – an den Web-Server zurück. Dieser leitet es weiter zum Browser. Benutzereingaben auf Basis von HTML-Formularen werden dem Web-Server über die Methoden `POST`, `PUT` bzw. `GET` des HTTP-Protokolls übermittelt. Abb. 4-4 zeigt, dass CGI-Programme außerhalb des Web-Servers ausgeführt werden. Das bedeutet, dass jeder Zugriff einen neuen Prozess startet, was hohe Antwortzeiten und eine starke Server-Belastung mit sich bringt. Ferner bedeutet dies, dass die Möglichkeit der Kommunikation zwischen unterschiedlichen Prozessen stark eingeschränkt und nicht mehr über den Web-Server steuerbar ist. Ebenfalls problematisch ist, dass die Sicherheitsaspekte seitens des Betriebssystems gelöst werden müssen und in der Regel nicht mehr über den Web-Server konfigurierbar sind.

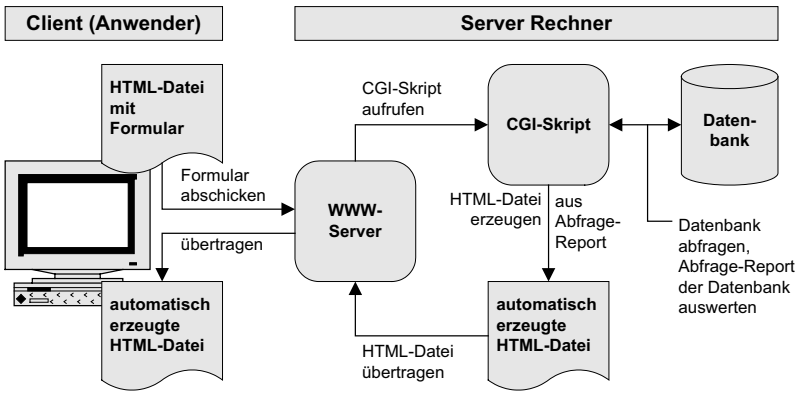


Abb. 4-4: *Eine mögliche Web/DB-Anbindung über CGI*

CGI-Programmierung stellt eine schnelle und flexible Möglichkeit der Web/DB-Anbindung dar. Insbesondere die Unabhängigkeit von der eingesetzten Programmiersprache sowie die Möglichkeit der Verwendung eines beliebigen Web-Servers und einer beliebigen Datenbank sind der Grund für die weite Verbreitung dieser Technologie. Bedingt durch den erforderlichen Prozessstart bei jeder Anfrage, das Zusammensetzen der Webseite zur Laufzeit und insbesondere die Notwendigkeit des Öffnens der Datenbank bei jeder Anfrage leidet die CGI-Technologie jedoch unter geringer Performanz. Auch das Neuladen einer Seite kann nicht vom Server identifiziert werden, weil er nicht erkennen kann, ob eine weitere Anfrage vom gleichen Client stammt oder ob es sich um einen neuen Client handelt. Bei einer auftretenden Verbindungsunterbrechung hält die Datenbank die Sitzung weiter offen, auch wenn die Browser-Verbindung schon lange unterbrochen wurde. Deshalb erscheint diese Lösung nur für WebIS mit geringem Datenvolumen, kurzen Prozesslaufzeiten und einer geringen Anzahl von Benutzerzugriffen empfehlens-

wert. Außerdem muss die durch die HTTP-Basiertheit gegebene Zustandslosigkeit aufwändig über bereits erwähnte Mechanismen umgangen werden.

Eine diesbezügliche Verbesserung stellt *FastCGI* dar (vgl. [FCGI02]), das das Offenhalten eines Prozesses über mehrere Anfragen hinweg erlaubt.

4.4.2 Web-Server API

Um die dem CGI immanenten Nachteile aufzuheben, wurden von verschiedenen Web-Server-Herstellern proprietäre Server-Erweiterungen vorgenommen (siehe auch [BeGr98]). Dabei ist es unter Ausnutzung des Application Programming Interface (API) des Web-Servers möglich, benutzerdefinierte Funktionen (Custom Functions) in den Server einzubringen. Die Funktionsangabe ist Bestandteil der URL und kann mit Parametern versehen werden. Neben den *nutzereigenen Funktionen* kann ein API auch *servereigene Funktionen* (Server Application Functions) unterstützen. Diese dienen der serverinternen Kommunikation, während nutzereigene Funktionen auch den Zugriff auf Datenbanken realisieren können. Sie werden nach dem Binden mit dem Server in dessen Adressraum eingebettet und sind kein eigenständiges Programm mehr. Der Start eines externen Programms für jeden Zugriff entfällt, weshalb DB-Verbindungen erhalten bleiben. Die zwei am meisten verbreiteten APIs sind das NS-API von Netscape [NSAPI] und das IS-API von Microsoft [ISAPI].

Das Verfahren bietet einige Vorteile. So sind die Sitzungen direkt im Web-Server verwaltbar. Die dauerhafte DB-Verbindung bringt auf jeden Fall Performanzvorteile. Zudem können die DB-spezifischen Operationen gezielt eingesetzt werden. Schließlich ist es möglich, das Zieldokument im HTML-Editor zu erstellen. Diesen Vorteilen stehen aber auch einige Nachteile gegenüber. Die Tatsache, dass Funktionen auf der Server-Seite eingebunden werden können, macht das Debugging wegen der Mischung aus lokaler und externer (serverseitiger) Kontrolle schwierig. Auch ist ein hoher Einarbeitungsaufwand in die API-Spezifikation notwendig. Offene DB-Transaktionen können nicht beendet werden, bleiben also hängen, falls auf der Client-Seite ein Fehler auftritt (z.B. Absturz des Browsers oder Rechners). Schließlich besteht eine gewisse Abhängigkeit von API-kompatiblen Web-Servern

4.4.3 Server Side Include (SSI)

Ein anderes Verfahren zur dynamischen Gestaltung von Webseiten sind die so genannten Server Side Includes (SSI). In Dokumenten enthaltene SSI-Befehle werden durch den Web-Server vor dem Versenden ausgeführt. Der Zugriff auf die DB erfolgt über vordefinierte und im Dokument eingebettete Kommandos. Die Abbildung unten zeigt ein Beispiel mit dem von Netscape verwendeten System LiveWire. Das Ergebnis einer einfachen SQL-Anfrage wird in einer Variablen gespeichert, welche dann weiter bearbeitet wird.

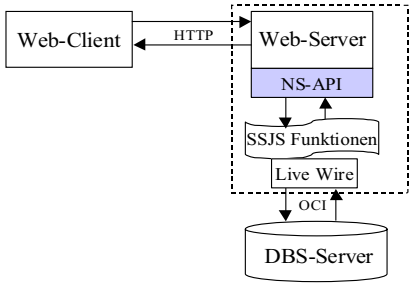


Abb. 4-5: *Datenbankanbindung über Server-Erweiterungen*

Abb. 4-5 zeigt als Beispiel die Architektur des auf NS-API aufsetzenden Produkts LiveWire. Programmfunktionalität wird auf Basis von Netscapes *Server-Side JavaScript* (SSJS) [SSJS98] realisiert. Die über das <SERVER>-Tag in HTML eingebetteten SSJS-Funktionen werden bei Anfrage der HTML-Seite ausgeführt und das Ergebnis eingebettet in den umgebenden HTML-Code an den Web-Client übermittelt (siehe Abb. 4.6).

```
<HTML>
...
<SERVER>
database.CONNECT(...)
result = database.cursor("SELECT name, age FROM Person")
name1 = result.name
age1 = result.age
document.WRITELN(name1 + ", " + age1 + "<BR>")
result.NEXT()
name2 = result.name
...
</SERVER>
</HTML>
```

Abb. 4-6: *SSI-Beispiel mit LiveWire*

Dieses Verfahren hat wie CGI den Nachteil, dass längerfristige Datenbankverbindungen nicht direkt unterstützt werden. Die Verbindung zur Datenbank besteht nur während des Parsens der Datei. Die Eigenschaften bezüglich der Transaktionssteuerung entsprechen jenen von CGI.

Vorteilhaft gegenüber dem CGI-Konzept ist die schnelle und leichte Programmierung der SSI-Kommandos. Sie stellen normalerweise keine Programme dar, sondern sind als Steuerungsbefehle über das Dokument verteilt. Ein möglicher Leistungsvorteil ergibt sich durch die Auslagerung der Datenbankzugriffe auf den Server. Allerdings hängt dieser Vorteil auch sehr von der Qualität der Implementierung ab. Ein Nachteil ist, dass der Web-Server SSI unterstützen muss. Es existieren erweiterte Formen von SSI, die jedoch von Server zu Server sehr unterschiedlich sind. Man sollte daher schon beim Erstellen der HTML-Dokumente

wissen, welchen Server man einsetzen wird bzw. welchen man zur Verfügung hat. Ein einheitlicher Standard für die erweiterten Formen von SSI wäre wünschenswert.

4.4.4 Active Server Pages (ASP)

Eine besondere Form von SSI stellen die *Active Server Pages* (ASP, [ASP01]) von Microsoft dar. Dies sind HTML-Dokumente, die eingebettete Anweisungen in VBScript (eine Teilmenge von Visual Basic) oder JScript (Microsofts Version von JavaScript) enthalten. ASP stellt eine Umgebung für *Skripte*, Anweisungsfolgen in einer Skriptsprache, dar und ermöglicht ihnen die Interaktion mit dem Web-Server. Die Skripte generieren auf Anforderung Elemente von Webseiten, insbesondere HTML-Code. Die ASP-Anweisungen werden verarbeitet und zusammen mit statischen Inhalten an den Browser gesendet. ASP ist keine eigenständige Applikation, sondern wird von Microsoft mit dem Internet Information Server ab Version 3.0 oder dem Personal Web-Server ab Version 4.0 ausgeliefert. Unter Microsoft Windows 2000 (Internet Information Server 5) ist ASP standardmäßig installiert, soweit die Web-Services (IIS) mitinstalliert wurden. Module für verschiedenste Anforderungen, wie z.B. ein Datenbankzugriff, sind bereits enthalten. Tatsächlich erfolgen Datenbankzugriffe über ActiveX Data Objects (ADO). Diese sind eine Sammlung von Komponenten, die von Microsoft entwickelt und auch in anderen Umgebungen als ASP eingesetzt werden. Session Management ist mit ASP sehr einfach. Es wird von ASP automatisch ein *Cookie* gesetzt, der einen Besucher der Website während einer Sitzung identifiziert. Der Cookie ist temporär, d.h., er wird gelöscht, wenn der Browser geschlossen wird. Probleme entstehen, wenn Cookies nicht unterstützt oder abgelehnt werden, da dann der Zugriff auf das Session-Objekt nicht mehr funktioniert. Die Verwendung von URL-Kodierung als Alternative ist nicht vorgesehen. Die ASP-Skriptsprachen besitzen nicht die Mächtigkeit einer normalen Programmiersprache wie Java oder C++.

Wie aus dem oben Gesagten hervorgeht, verfügen ASP über alle Möglichkeiten, einfach einen sitzungs- und damit verbindungsorientierten Dienst bereitzustellen. Prinzipiell wird dabei mit einer eindeutigen Session-ID gearbeitet, die mit Hilfe von Cookies übertragen wird (oder falls diese nicht verfügbar sind, mit einem Parameter SESSION-ID=... in der URL). Das funktioniert auch mit anderen Skriptsprachen. Bei ASP ist es allerdings besonders einfach anwendbar, da es hier fester Sprachbestandteil ist. Weitere Workarounds lassen sich auch über Cookies, versteckte Formularfelder (die die persistenten Parameter übertragen) oder serverseitige Komponenten (etwa ActiveX oder Java-Applets) erreichen. Bei Letzteren besteht zudem die Möglichkeit, eine zusätzliche TCP-Verbindung aufzubauen und den sitzungsspezifischen Datenaustausch über diese Verbindung zu handhaben bzw. zu koordinieren.

4.4.5 Java Servlet (Serverside Applet)

Servlets sind serverseitige Anwendungen, die dazu verwendet werden können, dynamisch Dokumente zu erzeugen und/oder Datenbankzugriffe zu regeln [JaSer99]. Sie werden auch häufig als serverseitige Applets bezeichnet, da sie einen einfachen und konsistenten Mechanismus zur Erweiterung der Funktionalität eines Web-Servers darstellen, allerdings ohne grafische Oberfläche. Voraussetzung für die Unterstützung von *Java Servlets* ist die Integration einer meist produktunabhängigen JVM in den Web-Server. Außerdem muss eine Servlet Engine eingerichtet werden, die als eine Art Plug-in auf der Server-Seite für die Ausführung der Servlet-Klassen verantwortlich ist. Anfragen werden in Abhängigkeit von der angeforderten URL entweder vom Web-Server direkt verarbeitet oder an die Servlet Engine weitergeleitet (siehe Abb. 4-7). Die Möglichkeit der Nutzung objektorientierter Konzepte sowie die Plattform- und Herstellerunabhängigkeit des von SUN Microsystems eingeführten Java Servlet API – im Gegensatz zu den proprietären APIs von Netscape und Microsoft – sprechen maßgeblich für den Einsatz von Java Servlets.

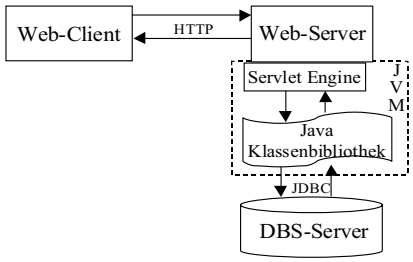


Abb. 4-7: *Web/DB-Anbindung über Java Servlets*

Durch die Verwendung von Java-Technologie können Servlets die volle Funktionalität der Java-Klassenbibliotheken verwenden. Besonders hervorzuheben ist die Netzwerk- und Datenbankunterstützung. Damit hat der Programmierer den Vorteil, alle Möglichkeiten der Sprache Java nutzen zu können. Für den Anwender liegt der Vorteil in einer verbesserten Abarbeitungsgeschwindigkeit. Das Sicherheitskonzept von Java gilt auch für Servlets. Die Servlets sind nicht auf HTTP-Server festgelegt, sondern können mit allen Servern arbeiten, die auf dem Anfrage/Antwort-Prinzip aufgebaut sind. Servlets können dabei die Anfragen entgegennehmen, sie bearbeiten und dynamisch erzeugte Antworten zurückgeben. Ein Servlet kann mehrere Anfragen parallel bearbeiten. Durch Verwendung der *Standard Java Extension API* funktionieren Servlets unverändert auf den verbreitetsten Web-Servern wie Netscape-NIS, Apache, Microsoft-IIS, oder Suns Java Web-Server.

Typische Anwendungsgebiete für Servlets sind:

- Auslieferung von Informationen aus einer Datenbank (sehr einfach durch sql-Package)
- Chats und Online-Konferenzen (mehrere Anfragen können parallel bearbeitet und synchronisiert werden)
- Bestellsysteme, die auf Produktdatenbanken basieren und den Lagerstand verwalten.

Servlet-CGI-Vergleich:

- *Leistung:* Servlets werden bei der ersten Anfrage initialisiert und bleiben nach der Abarbeitung im Speicher des Servers erhalten, während CGI-Prozesse immer neu gestartet werden. Das ist zum Beispiel für Applikationen, die auf eine Datenbank zugreifen, vorteilhaft, da die Verbindung zur Datenbank nur ein einziges Mal hergestellt werden muss.
- *Wiederverwendbarkeit:* Servlets können als Klassen importiert und an anderer Stelle verwendet werden. CGI wird meist für isolierte Anwendungen verwendet.
- *Sicherheit:* Der Security Manager von Java verhindert Ressourcenmissbrauch. CGI haben keine solche Funktionalität.
- *Lastverteilung:* Servlets können andere Servlets aufrufen. Dadurch kann u.a. die Lastverteilung gesteuert werden. Bei CGI ist dies nur mit sehr hohem Aufwand realisierbar.
- *Programmieraufwand:* Im Vergleich zur CGI- oder Server-API-Implementierung ist ein erhöhter Programmieraufwand notwendig, der jedoch durch die Existenz entsprechender Entwicklungsumgebungen und die Möglichkeit einer einfacheren Wartung wieder relativiert wird.

In Tab. 4-1 sind die wesentliche Eigenschaften von Servlets und Applets zusammengefasst und einander gegenübergestellt.

Servlet	Applet
Ohne GUI-Komponente	Mit GUI-Komponente
Java-Programme auf Server-Seite	Java-Programme auf Client-Seite
Ausführung in VM des Servers	Ausführung in VM des Clients (Browser)
Lokale Ressourcen des Servers können nicht genutzt werden	Lokale Ressourcen des Clients können nicht genutzt werden

Tab. 4-1: *Vergleich von Servlets and Applets*

4.4.6 Java Server Pages (JSP)

Eine Weiterentwicklung des SSI-Mechanismus sind die *Java Server Pages* (JSP), die von Sun Microsystems als eine Erweiterung des Servlet-Konzepts eingeführt wurden (siehe z.B. [JSP01] oder [Tura01]).

Ziel ist die Entwicklung portabler, dynamisch generierter Web-Anwendungen. JSP bieten die Möglichkeit, das statische HTML mit dynamischem Code zu mischen. In einer ersten Annäherung kann JSP als eine HTML-Seite angesehen werden, in die zusätzlicher Code eingebunden wurde, der Anwendungslogik umsetzt, um dynamischen Inhalt zu präsentieren. Die Anwendungslogik kann u.a. JavaBeans, JDBC-Objekte, Enterprise Java Beans (EJB) oder RMI-Objekte (Remote Method Invocation) adressieren, wobei alle diese Komponenten/Technologien einfach aus einer JSP-Seite heraus aufgerufen werden können.

Dabei kann der Programmierer zunächst ganz normal seine HTML-Seite entwerfen, wobei er dazu beliebige Werkzeuge zur Webseiten-Erstellung nutzen kann. Anschließend erst kann der Code für den dynamischen Anteil eingebunden werden. Ähnlich wie bei XML geschieht dies über spezifische Tags, die oft mit »<%« anfangen und mit »%>« enden. In diesem Beispiel einer JSP-Sequenz auf einer HTML Seite

```
...
Thanks for ordering      /* HTML
<I><%= request.getParameter("title") %></I>       /* JSP-Sequenz
```

wird »Thanks for ordering *Core Web Programming*« am Bildschirm ausgegeben, falls die folgende URL

```
http://host/OrderConfirmation.jsp?title=Core+Web+Programming:
```

abgesetzt wird.

Grundsätzlich gibt es drei verschiedene JSP-Konstrukte, die in die HTML-Seite eingebettet werden können: Skripting-Elemente (scripting elements), Direktiven (directives) und Aktionen (actions). Über *Skripting-Elemente* kann Java-Code spezifiziert werden, der Bestandteil des resultierenden Java Servlets wird. *Direktiven* erlauben es, die Kontrolle über die Gesamtstruktur des Servlets zu bewahren. *Aktionen* legen bereits bestehende Komponenten fest, die genutzt werden sollen, oder erlauben es, das Verhalten der JSP-Engine zu steuern.

Eine mit JSP-Konstrukten durchsetzte HTML-Seite wird normalerweise mit der Erweiterung *.jsp* an der Stelle als Datei abgelegt, wo auch eine normale HTML-Seite erwartet wird. Obwohl die Datei wie normales HTML und nicht wie ein Servlet aussieht, wird sie doch vor der erstmaligen Nutzung von der so genannten JSP-Engine in ein normales Servlet überführt, kompiliert und geladen. Danach erlauben viele Web-Server die Einführung eines Aliasnamens, der wie ein Verweis auf die ursprüngliche HTML-Seite aussieht, in Wirklichkeit aber auf das Servlet oder die JSP-Seite verweist.

Vorteilhaft an Java Server Pages ist die strikte Trennung des Designs und der Präsentation einer Seite von deren Inhalt bzw. Logik. Diese Trennung erlaubt einen wiederverwendbaren, komponentenbasierten Entwurf von Web-Anwendungen. Als Nachteil kann die notwendige Installation einer zusätzlichen JSP Engine angesehen werden.

4.4.7 Portlets

Ein *Portlet* ist ein (kleines) Java-Programm oder eine in einer anderen zulässigen Programmiersprache implementierte Komponente und ist als solche innerhalb eines (oder mehrerer) Portale eingebunden. Portlets ähneln damit den Servlets, nur dass ihre Ausführung innerhalb der darüber liegenden Portal-Umgebung erfolgt. Damit sind Portlets Bausteine, aus denen individualisierte Portal-Seiten hergestellt werden können. Dies geschieht im Rahmen einer *Aggregierung*, d.h. Auswahl von relevanten Portlets durch den Benutzer und Zusammenbau von deren Ergebnissen zu einer individualisierten Webseite. Dies bedingt auch, dass die Präsentation der Ergebnisse einer Portlet-Ausführung verschiedenen (benutzerspezifischen) Anpassungen (z.B. im Hinblick auf den Präsentationsaspekt) unterzogen werden können muss. Dies führt zu dynamischen, individualisierten Inhalten innerhalb der Portal-Seite zu speziellen Themenbereichen (z.B. ein Börsenbericht mit genau den Kursinformationen für Aktien, die aus Sicht des Benutzers für ihn relevant sind, oder ein Wetterbericht, der das Wetter in der vom Benutzer favorisierten Umgebung zeitgenau wiedergibt).

Ähnlich wie Servlets laufen Portlets in so genannten *Portlet-Containern* ab. Portlets stellen also eine mögliche Implementierung des dazugehörigen Portlet-API dar. Dieses kann wiederum am besten durch das Model-View-Controller-Konzept beschrieben werden. Es zerlegt die Portlet-Funktionalität in einen *Controller*, der den eingehenden Aufruf des API abfängt und diesen auf Befehle umsetzt, die das dazugehörige *Modell* umsetzen und dabei von den eigentlichen Anwendungsdaten und der Anwendungslogik abstrahieren und schließlich die Resultate in der gewünschten *Sicht* (view) darstellen.

Die Trennung von API und dessen Umsetzung erlaubt, dass die Implementierung an beliebiger Stelle abgespeichert sein kann, beispielsweise auch auf einem anderen Portlet-Server. Dies würde dann zu einer entfernten Ausführung des Portlets führen, erlaubt also die beliebige Wiederverwendbarkeit von Portlet-Implementierungen.

Der Inhalt eines Portlets wird auf dem dazugehörigen Portlet-Server erzeugt. Zu dessen Erstellung kann auch auf Datenbanken zugegriffen werden (vgl. [Bea00]). Das Portlet-Konzept erleichtert somit die Integration von Anwendungen. Die Portlets sind dabei entweder bereits im Portal-Produkt enthalten, stammen von Portal-Providern oder können selbst entwickelt werden.

Für die Entwicklung von Portlets stehen kommerzielle Entwicklungsumgebungen zur Verfügung wie beispielsweise der *Web Sphere Portal Server* (IBM) [WPS]. Es existieren aber auch Open-Source-Lösungen wie beispielweise *Apache Jetspeed*. Diese Systeme stellen meist bereits vorhandene Portlets zur Verfügung, die sehr allgemeine Funktionalität anbieten, auf die der Portlet-Entwickler zurückgreifen kann, wie z.B. ein Portlet für das Laden einer Webseite eines anderen Web-Servers in den eigenen Web-Server.

Oracle hat mit Portal 3.0 (vgl. [Ora00]), das auf der Servlet-Technologie basiert, einen neuen Standard für den Aufbau von E-Business-Portalen vorgestellt.

Hier bilden die Portlets die zentrale Schnittstelle zwischen dem Benutzer und den darzustellenden Daten.

Beim Portal von Bea ist die Benutzeroberfläche ebenfalls in Form von Portlets gestaltet worden (vgl. [Bea00]). Jedes Portlet besteht aus einer eigenen JSP-Seite, die sich themenbezogen selbst darstellt. Portlets kommunizieren untereinander mittels Pipelines. Geschäftsprozesse können für jedes Portlet hinterlegt werden. Sie triggern das Portlet und entscheiden anhand definierter Regeln, welche Informationen dem Benutzer des Portals angezeigt werden sollen.

4.5 Übergreifende Technologien

Neben den bereits diskutierten client- und serverseitigen Technologien gibt es noch einige weitere übergreifende Technologien, die insbesondere von den Erstgenannten zur Umsetzung ihrer Logik genutzt werden können. Dazu gehören als Java-basierte Zugriffstechnologien für DBS die Java Database Connectivity (JDBC) sowie SQLJ einerseits und andererseits mit Enterprise Java Beans (EJB) ein Rahmenwerk, das die Erzeugung von portierbaren, auf entsprechenden Servern vorgehaltenen und in der Programmiersprache Java implementierten Komponenten erlaubt. Diese drei Technologien sollen in diesem Abschnitt kurz vorgestellt werden.

4.5.1 Java Database Connectivity (JDBC)

JDBC (Java Database Connectivity API [JDBC99]) ist der bekannteste und am weitesten ausgereifte Ansatz zum Zugriff auf relationale DBS von Java-Programmen aus. Ähnlich der Philosophie von ODBC bietet JDBC ein API für den Datenbankzugriff, welches Basis-SQL-Funktionalität unterstützt und den Zugriff auf eine Vielzahl von DBMS-Produkten erlaubt. Auf der Basis von JDBC kann Java als die Hostsprache zur Entwicklung von Datenbankanwendungen genutzt werden. Auf dieser vergleichsweise niedrigen Schnittstelle können dann höherwertige Schnittstellen (wie z.B. SQLJ) aufsetzen.

JDBC ist eine Sammlung von Klassen, die im *java.sql*-Paket zusammengefasst sind. Diese Klassen realisieren SQL-Befehle, Ergebnismengen und Datenbankmetadaten. Die Schnittstellenspezifikation dieses (Low-level-)API zur Ausführung von SQL-Befehlen beruht auf dem *X/Open SQL CLI* (Call Level Interface), das auch die Grundlage der ODBC-Schnittstelle ist.

Das JDBC API stellt zwei wesentliche Schnittstellen bereit. Einerseits gibt es das API für den Anwendungsprogrammierer und andererseits das darunter liegende Treiber-API. Anwendungen benötigen diese Treiber, um auf Datenbanksysteme zugreifen zu können (siehe Abb. 4-8).

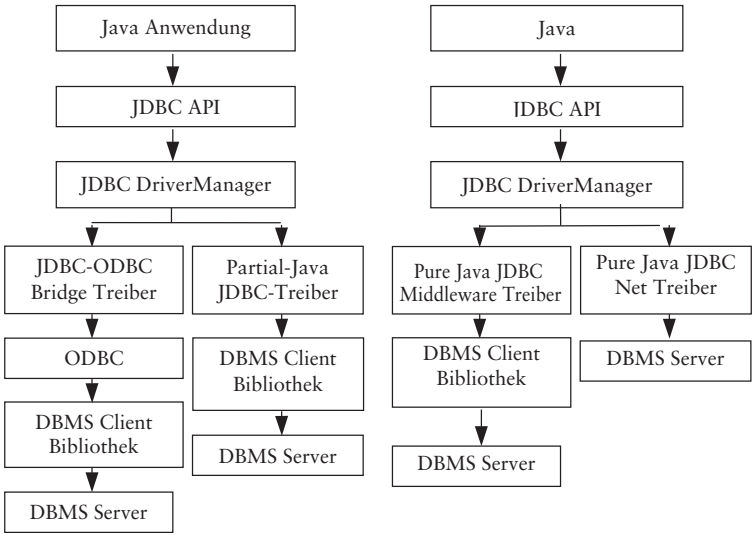


Abb. 4-8: *JDBC-Treiber und ihre Arbeitsweise*

JDBC-Treiber

JDBC nutzt einen Treibermanager (*java.sql.DriverManager*), der verschiedene Treiber unterstützt, die eine Verbindung zu verschiedenen Datenbanken aufbauen. JDBC kennt vier verschiedene Treibertypen:

- Typ 1: JDBC-ODBC-Bridge-Treiber
- Typ 2: Partial-Java-JDBC-Treiber
- Typ 3: Pure-Java-JDBC-Middleware-Treiber
- Typ 4: Pure-Java-JDBC-Net-Treiber

Die ersten beiden Typen erwarten, dass auf der Client-Seite spezifischer Code vorliegt, weshalb sich diese Varianten weniger gut für das Internet eignen. Beim ersten Typ wird aus JDBC heraus auf ODBC-Treiber zugegriffen, was bedeutet, dass die Java- und JDBC-Funktionalität auf die vergleichsweise schmalbrüstige ODBC-Funktionalität abgebildet wird. Dadurch geht einiges an Mächtigkeit von Java und JDBC verloren. Damit diese Variante funktioniert, muss clientseitig entsprechender ODBC-Binärcode geladen sein.

Die zweite Variante unterscheidet sich von der ersten nur dadurch, dass nicht der Umweg über ODBC gegangen wird, sondern direkt auf der Datenbank gearbeitet wird. Dazu muss dann eine entsprechende Software des spezifischen DBMS auf dem Client-Rechner installiert sein. Insgesamt ist diese Verbindung wegen des Direktzugriffes auf die DB sehr effizient.

Die letzten beiden Varianten unterstellen, dass der Treiber in reinem Java geschrieben und damit plattformunabhängig ist, also jederzeit dynamisch von einem Server- auf den Client-Rechner geladen werden kann. Deshalb eignen sich beide Varianten hervorragend für das Internet. Die häufigste Variante ist Typ 3, bei der JDBC-Anweisungen in ein middlewarespezifisches Protokoll übersetzt

werden, von wo aus sie auf eine oder mehrere DBS-Protokolle übersetzt werden. Die DBS werden also über die Middleware vom Client gekapselt, weshalb das Ersetzen eines DBS durch ein anderes die Client-Software nicht berührt.

Die vierte Variante ist schließlich wiederum eine, bei der direkt auf die DB zugegriffen wird. JDBC-Anweisungen werden direkt in das Netzwerkprotokoll des DBMS übersetzt und ausgeführt. Die direkte Verbindung zwischen Client und Datenbank-Server garantiert eine hohe Performanz.

Mit der Version 2 von JDBC wurde zusätzlich zu den Kernklassen in java.sql die Standarderweiterung javax.sql hinzugefügt. Diese bietet zusätzliche Fähigkeiten, zu denen neben Performanzverbesserungen auch die Unterstützung der neuen SQL:1999-Datentypen gehört. Tab. 4-2 zeigt eine Übersicht der neuen Methoden und Konzepte von JDBC 2.

Neue Methoden/Konzepte	Erweiterungen von Standard SQL
Scrollbare und änderbare ResultSets	Connection Pooling
SQL:1999-Typen	Verteilte Transaktionen
Batch Updates	Java Naming und Directory Interface (JNDI)
Package: java.sql	Package: javax.sql

Tab. 4-2: *Übersicht über JDBC 2*

JDBC versus ODBC

ODBC ist als DBS-unabhängige Schnittstelle weit verbreitet, eignet sich jedoch nicht zur direkten Verwendung aus Java-Programmen heraus, da es sich um ein C-Interface handelt. Aufrufe von C-Code aus Java würden jedoch die Sicherheit, Robustheit und leichte Portierbarkeit eines Programms beeinträchtigen, alles Eigenschaften, die bei der Entscheidung für Java als Programmiersprache üblicherweise wesentlich sind.

Als Vorteile beim Einsatz von JDBC sind zu nennen die gegebene Plattformunabhängigkeit und die einfache Programmierung, die zu kurzen Entwicklungszeiten führt. Nachteilig wirkt sich aus, dass Fehler erst zur Laufzeit erkannt werden. Zudem leidet die Performanz, da die übersetzten Programme zur Laufzeit interpretiert werden müssen. Lösungen hierzu sind JITs (Just-in-Time-Compiler) und optimierte JVMs.

4.5.2 SQLJ

SQLJ ist ein von vielen führenden Datenbankherstellern (wie Oracle, IBM, Sybase, Tandem und Sun Javasoft) entwickelter Standard, der es erlaubt, SQL in Java-Programme einzubetten. Im Gegensatz zu JDBC, das die dynamische Ausführung von SQL-Code erlaubt, unterstützt SQLJ nur die statische Einbettung. Dies hat den Vorteil, dass bereits zur Übersetzungszeit die SQL-Anweisungen auf

Typkonsistenz überprüft werden können und auch eine Optimierung vorgenommen werden kann. Der SQLJ-Vorübersetzer wandelt SQL-Anweisungen in Standard-Java-Code um, der es erlaubt, auf die Datenbank über ein CLI zuzugreifen.

SQLJ besteht aus drei Teilen:

- *Part 0: Embedded SQL*: Dieser Teil beschreibt, wie statische SQL-Anweisungen in Java eingebettet werden.
- *Part 1: Java Stored Procedures:* Dies sind Prozeduren und Funktionen, die im Datenbank-Server gespeichert und ausgeführt werden und damit lokalen Zugriff auf Daten haben.
- *Part 2: Java-Klassen für SQL-Datentypen:* Teil 2 des SQLJ-Standards beschreibt, wie Java-Klassen zur Implementierung von SQL-Datentypen benutzt werden.

SQLJ und JDBC sind interoperabel, d.h., SQLJ und JDBC-Anweisungen können miteinander gemischt eingesetzt werden.

Die SQLJ-Anweisungen beginnen in Java-Quelltexten mit dem Schlüsselwort #sql, darauf folgen in der Regel SQL-Anweisungen. Die SQLJ-Programme werden durch einen Vorübersetzer, *SQLJ-Translator*, aufbereitet. Aus den SQLJ-Quelltexten werden Java-Quelltexte generiert. Der Übersetzer kann bereits zur Übersetzungszeit Prüfungen der SQL-Befehle durchführen. Neben der SQLJ-Syntax wird auch die SQL-Semantik geprüft. Beispielweise wird geprüft, ob die Befehle zum Schema der Datenbank passen. Aus den SQL-Anweisungen werden Aufrufe des SQLJ-Laufzeitsystems generiert.

SQLJ stützt sich in seiner Implementierung auf JDBC, d.h., man kann SQLJ als eine High-level-Schnittstelle zur Datenbank ansehen, die auf der Low-level-Schnittstelle JDBC aufsetzt. Tab. 4-3 fasst die wesentlichen Merkmale beider Technologien noch einmal zusammen.

Zu den Vorteilen von SQLJ gehört, dass SQLJ-Programme vergleichsweise kompakt und daher gut lesbar sind. Weiterhin ist eine höhere Fehlersicherheit gegeben, da durch die Überprüfung von SQL-Anweisungen schon während der Übersetzungsphase Typ- und Schemainkompatibilitäten sowie Syntaxfehler festgestellt werden können. Zudem kann eine Optimierung vorgenommen werden. Schließlich ist durch die JDBC-Basis die Plattform- und Datenbanksystem-Unabhängigkeit gewährleistet. Als nachteilig ist anzusehen, dass SQL-Anweisungen nicht zur Laufzeit erzeugt werden können. Auch ist der Java-Code, den der Übersetzer erzeugt, schwer lesbar. Dies erschwert das Debugging. Schließlich dauert auch der Übersetzungsvorgang länger.

JDBC	SQLJ
Call Level Interface (CLI)	Embedded-SQL Lösung, die auf einem CLI aufsetzt.
dynamisches SQL	statisches SQL
Fehlererkennung zur Laufzeit	Fehlererkennung zur Übersetzungszeit

Tab. 4-3: *Vergleich von SQLJ and JDBC*

4.5.3 Enterprise Java Beans (EJB)

Bei E*nterprise* J*ava* B*eans* (EJB) handelt es sich um ein von Sun Microsystems vorgeschlagenes serverbasiertes Rahmenwerk, das die Erzeugung von portierbaren, auf entsprechenden Servern vorgehaltenen und in der Programmiersprache Java implementierten Komponenten erlaubt (vgl. z.B. [LiNa99]). Zunehmend etabliert sich EJB als Industriestandard und fördert dadurch die Entstehung eines Komponentenmarktes. Die auf Basis der EJB-Spezifikation entwickelten Komponenten werden als *Enterprise Beans*, Instanzen dieser Komponenten als *Enterprise-Objekte* bezeichnet. Als Grundlage der Kommunikation zwischen Enterprise-Komponenten und Clients dient die *Remote Method Invocation* (RMI), die Sun-spezifische Variante des Remote Procedure Calls (RPC). EJB ist Bestandteil der *Java 2 Enterprise Edition* (J2EE), einer Spezifikation, die eine Vielzahl von APIs miteinander vereint. Bestandteil dieser Spezifikation ist eine Architekturbeschreibung für Internet-Anwendungen. Abbildung 4-9 illustriert ein typisches Szenario einer J2EE-Applikation. Clientseitig wird durch den Browser bzw. durch Applets via HTTP auf einen Web-Server zugegriffen. Dieser wiederum verwendet die J2EE-APIs, um mit serverseitigen Komponenten zu kommunizieren.

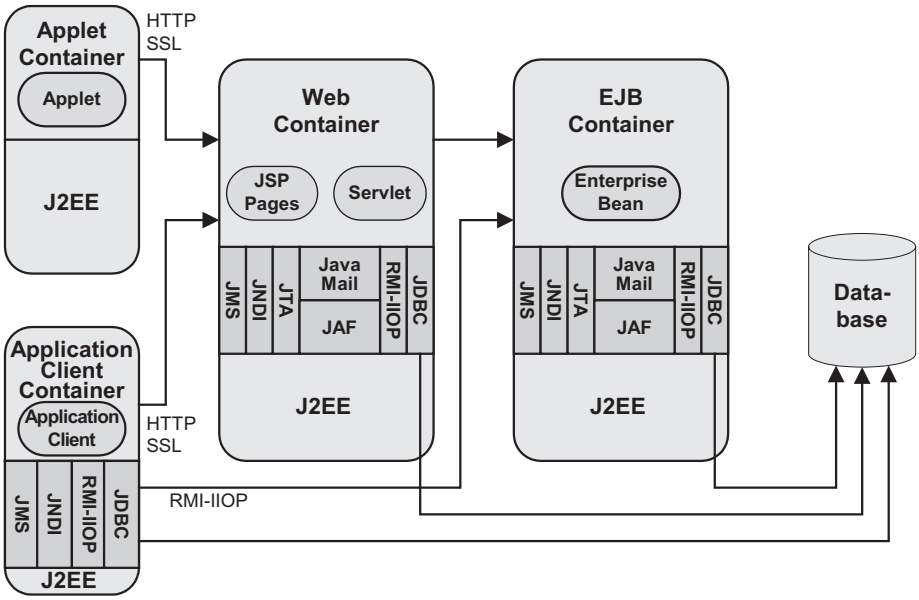


Abb. 4-9: *EJB und der Zugriff auf Datenbanksysteme*

Wie bei objektorientierter Middleware üblich, haben auch EJB-Komponenten über ein vorgegebenes Java-API Zugriff auf notwendige Dienste, wie Namens-, Persistenz- und Transaktionsdienst.

Die Instanziierung, Ausführung und Löschung von Enterprise Beans erfolgt in einem so genannten *EJB-Container*. Er abstrahiert von der konkret vorliegenden Hard- und Systemsoftware, indem er eine rechner- und betriebssystemunabhängige Laufzeitumgebung bereitstellt. Ferner ermöglicht EJB die für den Entwickler transparente Nutzung eines Persistenzdienstes. In diesem Fall ist der EJB-Container für die Zusicherung der Persistenz zuständig. Für die Integration von Legacy-Systemen erlaubt EJB ferner die Nutzung von Low-level-APIs (wie z.B. JDBC) durch den Entwickler. Die Aufgabe des EJB-Containers beschränkt sich in diesem Fall auf die Aktivierung und Synchronisation der serverseitigen Objekte.

Vorteilhaft wirkt sich auf jeden Fall die Trennung von Präsentation, Geschäftsprozesslogik und Daten aus. Auch sind im Durchschnitt kürzere Entwicklungszyklen zu erwarten. Schließlich ist die Geschäftsprozesslogik plattform- und herstellerunabhängig implementierbar. Als Nachteil ist zu nennen, dass bei einer so jungen Technologie wie EJB der Standardisierungsprozess noch nicht abgeschlossen ist. Auch gibt es ein stark unterschiedliches Leistungsspektrum zwischen Servern.

4.6 Klassifikation von WebIS

WebIS weisen unterschiedlichste Komplexitätsgrade auf. Sie können einfache Präsentationen darstellen, beispielsweise einer Firma, aber auch vollständige E-Commerce/E-Business-Anwendungen umsetzen. Im Folgenden werden anhand der Entwicklungshistorie und verschiedener Architekturmerkmale fünf verschiedene Varianten von WebIS identifiziert und deren typische Einsatzgebiete bzw. Stärken und Schwächen diskutiert. Jüngere Architekturalternativen weisen dabei i.d.R. eine höhere Flexibilität (im Sinne der dynamischen Einbindung und Gestaltung von Inhalten) auf, was aber nicht gleichzusetzen ist mit einer Ersetzung der älteren Generation durch eine neuere. Vielmehr haben nach wie vor alle Architekturvarianten ihre spezifischen vorteilhaften Anwendungsgebiete, in denen sie als die optimale Lösung angesehen werden können. Dementsprechend vereinen in der Realität insbesondere komplexe WebIS typischerweise mehrere der vorgestellten Architekturalternativen, indem sie für jeden Bereich des Gesamtsystems die jeweils geeignete Architektur einsetzen.

4.6.1 Variante 1: Statische WebIS

Statische WebIS stellen die originäre Generation eines WebIS dar. Webseiten werden dabei in Form von vorgefertigten, also statischen HTML-Dateien am Web-Server abgelegt. Im Falle einer Anfrage auf Basis des HTTP-Protokolls werden solche statischen Webseiten als Antwort an den Web-Client geschickt (siehe Abb. 4.1). Die Einführung des Common Gateway Interface ([CGI01], siehe Abschnitt 4.4) und von HTML-Formularen erlauben dem Benutzer eine erste einfache Form der *Interaktivität* über *Eingabebereiche*, *Radio-Buttons* oder *Aus-*

wahllisten. Durch die ausschließliche Verwendung von HTML zur Repräsentation der Webseiten werden *Inhaltsaspekte* (d.h. die eigentlichen Daten einer Webseite) mit *Hypertextaspekten* (d.h. der Aufbau einer Webseite inklusive Navigationsstrukturen) und *Präsentationsaspekten* (d.h. die grafische Repräsentation einer Webseite) vermischt. Dadurch wird eine *Änderungslokalität* im Sinne einer isolierten Modifikation jedes einzelnen Aspekts verhindert. Die Aktualisierung der Webseiten erfolgt häufig manuell über entsprechende Editoren und Werkzeuge. Dies stellt insbesondere bei änderungsintensiven Webseiten einen *kostenintensiven Faktor* dar und führt wegen der manuell nachzufahrenden Änderung von Realweltgegebenheiten häufig zu *veralteten Informationen*. Darüber hinaus besteht die Gefahr von *Inkonsistenzen*, da oftmals zum Zwecke eines komfortablen Informationszugangs bestimmte Inhalte redundant auf verschiedenen Webseiten bzw. auch an anderen Stellen innerhalb eines Unternehmens gespeichert werden. Der hauptsächliche Vorteil dieser Architektur liegt in ihrer *Einfachheit* und *Stabilität* sowie den *niedrigen Antwortzeiten*, da die Webseiten bereits vorgefertigt am Web-Server vorliegen.

Abb. 4-10: *Architektur eines statischen WebIS*

Die Entscheidung für ein statisches WebIS ist insbesondere bei Vorliegen folgender Voraussetzungen sinnvoll:

- geringe Anzahl von Webseiten,
- niedrige Änderungsfrequenz,
- Heterogenität der Webseiten hinsichtlich Hypertextstruktur und Präsentation.

4.6.2 Variante 2: WebIS mit DBS-Unterstützung

Mit Zunahme der in einem WebIS verwalteten Datenmenge wird die Verwendung eines DBS zur effizienten und konsistenten Verwaltung dieser Daten unumgänglich. Die Datenbankanbindung kann dabei, wie in Abbildung 4-11 durch gestrichelte Pfeile gekennzeichnet, einerseits auf der Server-Seite beispielsweise mittels CGI oder Java Servlets erfolgen oder andererseits auf der Client-Seite, z.B. mittels Java Applets. Die durch die Integration eines DBS erreichte Trennung zwischen Inhalts- und Präsentationsaspekten realisiert Änderungslokalität und vereinfacht damit die Aktualisierung von Webseiten (vgl. [ACPT99]). Ein weiterer Vorteil liegt im *hohen Grad* an *Interaktivität*. Zum einen kann dem Benutzer auf Basis von DB-Abfragen eine *strukturierte Suche* angeboten werden. Zum anderen besteht die Möglichkeit, durch Nutzung der Mechanismen zur Mehrbenutzerkontrolle eine *dezentrale Aktualisierung des Datenbestandes* auf Basis eines Extranets

zu erlauben (vgl. z.B. [PrRW01]). Die *Aktualität* der Daten, die *Stabilität* des Systems sowie die *Antwortzeiten* sind in hohem Maß davon abhängig, welche Strategie zur Generierung von Webseiten aus DB-Daten verwendet wird. Dabei können drei zueinander orthogonale Dimensionen unterschieden werden, nämlich der *Generierungszeitpunkt* (Zeitpunkt einer Anfrage versus Zeitpunkt einer Datenänderung), die *Materialisierungsart* (Dateisystem oder Cache-Bereich am Server/Client/Proxy – siehe Kapitel 7) und die *Generierungsgranularität* (System-Seiten- oder Attributebene [SKRP01]).

Ein Nachteil dieser Architektur liegt darin, dass durch die separate Speicherung des Inhalts der Webseiten entsprechende Abbildungsvorschriften verwaltet werden müssen, die letztendlich für die Zusammenstellung der Webseite verantwortlich sind. In Abhängigkeit von der gewählten Technologie zur DB-Anbindung (siehe Kapitel 5.2) sind diese Abbildungsvorschriften im Programmcode enthalten (HTML-generierende Applikation), oder die Abbildungsvorschriften entsprechen einer um Programmcode angereicherten HTML-Datei (HTML-Erweiterung).

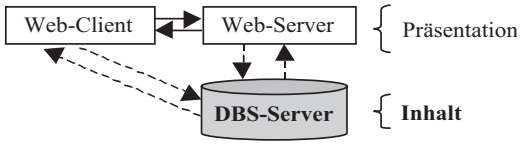


Abb. 4-11: *Architektur eines WebIS mit DBS-Unterstützung*

Folgende Merkmale, die z.B. bei der Präsentation umfangreicher Produktkataloge am Web gegeben sind, sprechen für die Integration eines DBS in ein WebIS:

- große Anzahl an Webseiten,
- hohe Änderungsfrequenz,
- Homogenität der Webseiten hinsichtlich Hypertextstruktur und Präsentation,
- Nutzung existierender (Legacy) Datenbestände.

Eine Weiterführung dieser Variante stellt die *Integration heterogener verteilter Daten* in ein WebIS dar. Die Motivation dafür liegt in dem Umstand, dass zum einen ein WebIS möglichst umfassende Information bieten soll, es aber keinen Sinn macht, die gesamte Information im lokalen DBS zu verwalten – nicht zuletzt deshalb, weil die Datenakquisition und -wartung eine nicht zu vernachlässigende Aufgabe darstellt. Zum anderen liegen umfangreiche Datenbestände in verteilten Quellen vor und sind über das Internet zugänglich. Informationen können dabei in *strukturierter* Form, z.B. in verteilten DBS, in *unstrukturierter* Form, z.B. als Textdokumente, oder in *semistrukturierter* Form, z.B. als HTML-Dateien in einem anderen WebIS, vorliegen. Die Integration dieser heterogenen, verteilten Daten wird dabei über Mediatoren- und Wrapper-Mechanismen realisiert (vgl. Kapitel 6). Diese ermöglichen sowohl den Zugriff auf die dezentrale Information – eingebettet in die Navigations- und Suchlogik des WebIS im Sinne eines »Single

Point of Access« – als auch die Verknüpfung lokaler und verteilter Daten in strukturierten Abfragen. Beispielsweise kann ein Web-basiertes Tourismusinformationssystem Information über Orte und Hotels in einem lokalen DBS speichern, während es Zusatzinformation, wie einen Veranstaltungskalender, von einem anderen WebIS abruft [FGLM98]. Die Problematik dieses Ansatzes liegt darin, dass diese Konzepte im Allgemeinen auf Präsentationsaspekten der dezentralen WebIS aufbauen und Änderungen derselben sowohl ein entsprechendes Benachrichtigungskonzept erfordern als auch Anpassungsarbeiten nach sich ziehen.

4.6.3 Variante 3: Applikationsorientierte WebIS

Die zunehmende Komplexität von WebIS sowie die permanent wachsende Anzahl an Benutzern führen häufig zu Stabilitätsproblemen sowie mangelnder Performanz. Die dritte Variante von WebIS versucht diese Probleme zu lösen, indem Anwendungslogik nicht mehr am Web-Server, sondern auf einem oder mehreren Applikations-Servern gespeichert und ausgeführt wird (siehe Abb. 4-12). Der Web-Server ist dabei nur mehr für das Weiterreichen der Anfragen von Clients an Applikations-Server sowie für den Transfer des vom Applikations-Server gelieferten Ergebnisses (z.B. HTML-Seiten) an den Client zuständig. Die Funktionalität von Applikations-Servern beschränkt sich nicht alleine auf das Zusammenstellen der HTML-Seite und die Ausführung von Applikationslogik, sondern umfasst insbesondere auch DB-Anbindung, Transaktionsmanagement, Sicherheitsvorkehrungen sowie Lastausgleich und Caching. Analog zu der in Abschnitt 4.3 diskutierten Möglichkeit einer clientseitigen Anbindung einer Datenbank, kann ein Client auch ohne Umweg über den Web-Server eine direkte Verbindung zum Applikations-Server aufnehmen. Dies erfolgt meist auf Basis eigener Protokolle wie z.B. IIOP, wodurch die Nachteile des HTTP-Protokolls aufgehoben werden. Der Web-Server ist dabei nur für das Zustandekommen der initialen Netzwerkverbindung via HTTP zuständig.

Der Funktionsumfang kommerzieller Applikations-Server ist sehr unterschiedlich. Dabei können prinzipiell kombinierte Web-/Applikations-Server[1] von Enterprise-Application-Servern[2], die umfangreiche Gesamtlösungen bieten, unterschieden werden [Mari00].

Ein schwerwiegender Nachteil dieser Systeme ist die meist gegebene Abhängigkeit von einem bestimmten Anbieter, die außerdem massiv das Entwicklungsumfeld bestimmt und insbesondere bei den oben genannten Gesamtlösungen mit hohen Kosten für die Basissoftware verbunden ist.

Eine auf ein applikationsorientiertes WebIS ausgerichtete Architektur ist empfehlenswert, wenn folgende Merkmale gegeben sind:

1. z.B. *Coldfusion Web Application Server* (Allaire Corp.), *Netscape Application Server* (Netscape Communication Corp.), *Oracle Internet Application Server* (Oracle Corp.)
2. z.B. *WebSphere Application Server* (IBM) und *WebLogic* (Bea Systems)

- Hohe zu erwartende Server-Last, wie sie durch eine große Anzahl gleichzeitig zugreifender Clients entsteht.
- Komplexe Geschäftslogik, wie sie z.B. WebIS aufweisen, die einen Online-Kauf oder eine Online-Buchung erlauben.
- Hohe Transaktionsorientiertheit, wie z.B. bei Online-Banking-Anwendungen, wobei hier der Aufbau einer direkten Client/Server-Kommunikation sinnvoll zu sein scheint.

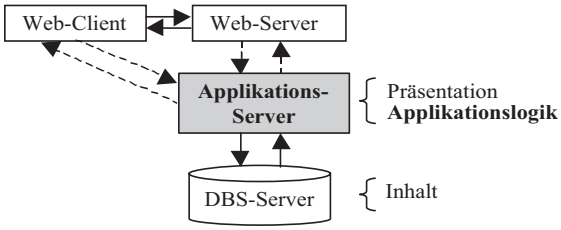


Abb. 4-12: *Architektur eines applikationsorientierten WebIS*

4.6.4 Variante 4: Ubiquitäre WebIS

Die zunehmend an Bedeutung gewinnende vierte Variante von WebIS sind so genannte *ubiquitäre WebIS*. Ein ubiquitäres WebIS stellt personalisierte Dienste zu jeder Zeit an jedem Ort und für jedes sinnvoll nutzbare Medium zur Verfügung, womit ein allgegenwärtiger Zugriff möglich wird. Dabei ist das oberste Ziel, *den richtigen Dienst* zum *richtigen Zeitpunkt* am *richtigen Ort* in der *richtigen Form* anzubieten. Ein Beispiel wäre die Anzeige von Mittagsmenüs auf den mobilen Endgeräten jener Benutzer, die zwischen 11.00 und 14.00 Uhr das Restaurant betreten. Eine wesentliche Voraussetzung dafür ist die Kenntnis des *Kontexts,* in dem das WebIS gerade benutzt wird, um bei Änderungen dieses Kontextes dynamisch, d.h. zur Laufzeit, entsprechende *Anpassungen* an verschiedenen Aspekten des WebIS vornehmen zu können. Wie in Abb. 4-13 dargestellt, sollte idealerweise die Kontextüberwachung, aber auch die Anpassung selbst von einer eigenen *Anpassungskomponente* übernommen werden. Letztere kann beispielsweise mit Hilfe eines Regelwerks (Ereignis-Bedingung-Aktion) die Anpassung der Inhalte bzw. der Präsentation der Information steuern.

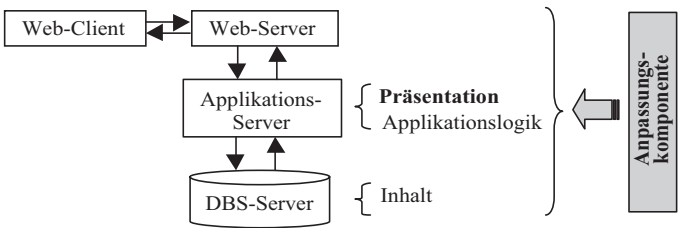


Abb. 4-13: *Architektur eines ubiquitären WebIS*

Insbesondere um Codeverdopplung im Hinblick auf Anpassungen zu vermeiden, muss eine klare Trennung der verschiedenen Aspekte eines WebIS gegeben sein. Dies kann durch den Einsatz von XML und XSL bzw. durch Anwendung des *Model-View-Controller-Prinzips* erreicht werden. Eine Realisierungsalternative stellt Apaches XML Publishing Framework Cocoon [Spie01] dar, das zur Trennung von Inhaltsaspekten, Präsentationsaspekten und Geschäftslogik so genannte XML-basierte eXtensible Server Pages (XSP) einsetzt (die große Ähnlichkeit zu JSP aufweisen) und als Integrationstechnologie XSL verwendet. Eine weitere Alternative wäre die Kombination der Technologien JSP (vgl. [JSP01]) für die Präsentation (View), Java-Servlets für die Steuerungskomponente (Controller) und Enterprise Java Beans für den Inhalt (Model). Der *Controller* ist für die korrekte Behandlung, Kontrolle und Verwaltung von Anfragen eines Clients zuständig. Die eigentliche Anforderung des Clients, also der Inhalt, wird über Enterprise Java Beans umgesetzt. Sie bedienen sich bei der Ausführung ihrer Dienste des EJB-Servers. Die Präsentationsaspekte der Ergebnisse werden schließlich über JSP als View-Komponente gesteuert (siehe Abb. 4.2).

Existierende WebIS dieser Variante bieten häufig nur eine sehr eingeschränkte Form der *Ubiquität*, indem beispielsweise nur Personalisierung oder lokationsbasierte Dienste unterstützt werden. Darüber hinaus wird meist sowohl die Kontextüberwachung als auch die Anpassung selbst mit kontextunabhängiger Funktionalität vermischt, was wiederum die Wartung erschwert. Beispiele für kommerzielle Systeme, die eine getrennte Anpassungskomponente realisieren, sind der Websphere Transcoding Publisher (IBM) [WTP] und die Oracle Wireless Edition (Oracle Corp.) [OWE].

Ubiquitäre WebIS eignen sich insbesondere bei Vorliegen folgender Anforderungen:

- Zugriff auf die im WebIS präsentierte Information nicht nur über das World Wide Web, sondern z.B. auch über mobile Endgeräte
- Anpassung der Inhalte und/oder der Präsentationsaspekte an Benutzerprofile
- Realisierung lokations/zeitabhängiger Dienste, wie z.B. ein lokationssensitiver Museumsführer

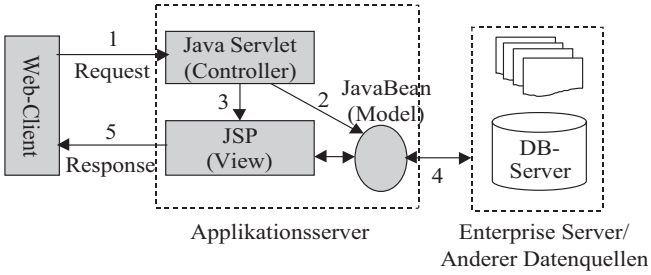


Abb. 4-14: *Das Model-View-Controller-Prinzip auf der Basis von EJB*

4.6.5 Variante 5: Portal-orientierte WebIS

Portale stellen die jüngste Variante von Web-Architekturen dar. Diese erlauben es einem Endbenutzer, aus einer Reihe von durch einen *Portal-Server* angebotenen *Portlet-Implementierungen* relevante auszuwählen und diese zu einem Portal zu aggregieren.

Um nicht nur lokale Portlets verwenden zu können, die von einem bestimmten Portal-Server bzw. einer Portlet-Entwicklungsumgebung angeboten werden, sondern auch so genannte »*Remote*« *Portlets*, wird versucht, Portlets und *Web Services* zu integrieren (vgl. z.B. [WSPR] und Kapitel 10 in diesem Buch). Web Services stellen dabei im Wesentlichen drei Basistechnologien zur Verfügung: (1) *Web Services Definition Language (WSDL)*, eine Sprache zur Schnittstellenbeschreibung, (2) *Simple Object Request Protocol (SOAP)*, ein XML-Protokoll für den verteilten Methodenaufruf und (3) *Universal Description, Discovery, and Integration (UDDI)* für die Benennung und das Auffinden verteilter Objekte [WSA]. In der WSDL beschriebene und vorher über UDDI aufgefundene Dienste können rekursiv zu komplexeren Diensten zusammengesetzt werden. Damit lassen sich durch Aggregation beliebig komplexe, individuelle Webseiten erstellen und anzeigen. Mit ihrer hohen Flexibilität und Dynamik decken Portale und Portlets auch sehr anspruchsvolle kommerzielle Anwendungen ab, wobei ein sehr hoher Grad an Individualisierung erzielt werden kann. Abstrakt gesehen erinnert die Portal-Technologie an die Komponententechnologie. Es werden aus einer Menge vorgefertigter Portlet-APIs die richtigen ausgesucht (inklusive der gewünschten Implementierungen), evtl. Anpassungen vorgenommen und dann die gewünschte, hoch individualisierbare Portal-Seite daraus erzeugt.

Performanzprobleme werden sicherlich immer dann zu erwarten sein, wenn auf entfernt abgelegte Portlet-Implementierungen zugegriffen wird. Hier wird als Lösung bereits angeboten, Ergebnisse von Portlet-Ausführungen lokal (z.B. in einer Datenbank) abzulegen und zunächst auf diese zuzugreifen. Das Problem ist hier aber die Aktualität der Daten und die Frage, wie diese erkannt bzw. sichergestellt werden kann.

Zudem gilt für die Portal-Variante ebenso wir für alle anderen Varianten, dass die Sicherheit ein sehr schwer wiegendes und weitgehend noch ungelöstes Problem darstellt. Dies wird ein wesentlicher Hemmschuh für die schnelle Verbreitung dieser Architektur sein.

Abb. 4-15 skizziert noch einmal die Trennung der Ebenen Inhalt, Applikationslogik und Präsentation bei der vorgeschlagenen Architektur für Portal-basierte WebIS. Diese Trennung war bereits ausführlich unter den Portalen in Abschnitt 4.4 diskutiert worden.

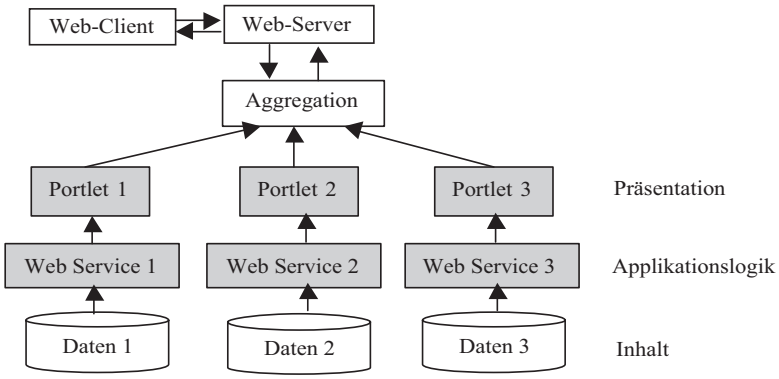


Abb. 4-15: *Architektur eines Portal-basierten WebIS*

4.6.6 Zusammenhang Technologien und Architekturen

Die Entscheidung für eine der angeführten Web/DB-Anbindungstechnologien ist in hohem Maß von den Anforderungen und Randbedingungen des jeweiligen Anwendungsbereichs abhängig. Abb. 4-16 präsentiert Beispiele für Web/DB-Anbindungstechnologien und schlägt damit auch die Brücke zu den ersten vier der in Kapitel 4.6 vorgestellten Architekturvarianten eines WebIS.

① zeigt ein statisches WebIS (Variante 1) mit Zugriff auf HTML-Seiten. ② bis ⑥ stellen alternative Technologie für die Umsetzung von WebIS der Variante 2 dar. Bei ② erfolgt ein Zugriff auf die Datenbank über CGI, wobei jede HTML-Seite über ein CGI-Programm zusammengesetzt wird. Bei ③ liegen Java-Applets auf dem Web-Server, die bei einer Anfrage zum Web-Client transferiert und dort ausgeführt werden. Hierbei erfolgt der Datenbankzugriff vom Client aus. ④ zeigt serverseitige API-Erweiterungen, die den Datenbankzugriff und das Zusammensetzen der HTML-Seite durchführen. Eine Servlet Engine und eine Java Virtual Machine sind in ⑤ für Java Servlets zu integrieren. Bei ⑥ (Variante 3) ist ein Applikations-Server zur Ausführung komplexer Applikationslogik zwischengeschaltet. Dieser nimmt auch die Verbindung zur Datenbank wahr. Datenbankinhalte können dabei aus Performanzgründen am Applikations-Server zwischengespeichert werden. ⑦ (Variante 4) stellt die Trennung der Präsentationsebene dar, indem z.B. XML/XSL-Technologie eingesetzt wird. Hier können die HTML-Dateien von einem Präprozessor zusammengebaut werden, bevor sie zu einem nicht XML-fähigen Web-Client transferiert werden.

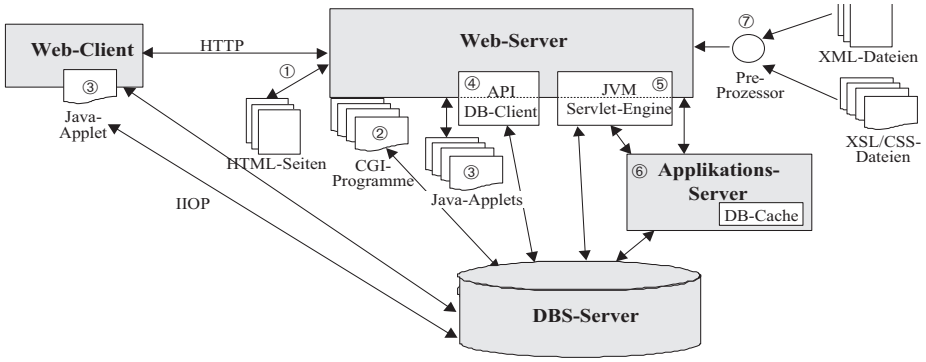


Abb. 4-16: *Technologien der Web/DB-Anbindung und WebIS-Architekturen*

4.7 Zusammenfassung

In diesem Kapitel wurden verschiedene Architekturen für WebIS vorgestellt. Für die Realisierung dieser Architekturen kann eine Vielzahl verschiedener Basistechnologien eingesetzt werden. Der Fokus wurde dabei zunächst auf Technologien zur Datenbankanbindung gelegt. Dabei wurden sowohl clientseitige Technologien, wie Java Applets und ActiveX, als auch serverseitige Technologien, wie Java Servlets, Server-API-Ansätze oder Java Server Pages, sowie übergreifende Technologien, wie JDBC, SQLJ und Java Enterprise Beans, vorgestellt.

Im zweiten Teil des Kapitels wurde eine an der Entwicklungshistorie orientierte Klassifikation von Architekturen vorgenommen. Ausgehend von statischen WebIS zur reinen Informationsbereitstellung entwickelten sich durch Integration von DBS und Applikations-Servern Architekturen, die den Anforderungen komplexer Softwareanwendungen genügten. Aktuelle Architekturen zielen insbesondere auf die Entwicklung kontextabhängiger Dienste in Form von ubiquitären WebIS und der Aggregation von Diensten in Form von Portalen ab. Die Präsenz dieser Architekturen in einem konkreten WebIS ist dabei nicht ausschließend. Vielmehr integrieren reale WebIS mehrere Architekturen, um deren Vorteile für einen bestimmten Bereich nutzen zu können.

Literatur

[ACPT99] Atzeni P., Ceri S., Paraboschi S., Torlone R. *Database Systems: Concepts, Languages & Architectures*. McGraw Hill, 1999.

[Aper94] Apers P. *Identifying Internet-related Database Research*. Second International East-West Database Workshop. Extending Information Systems Technology. ed: J. Eder, L. Kalinichenko. Klagenfurt, Austria, Sept. 1994.

[ASP01] *Active Server Pages (ASP) Technology Feature Overview.* Microsoft Corporation. http://msdn.microsoft.com/library/default.asp?URL= library/psdk/iisref/aspguide.htm (19 August 2001).

[BCLN94] Berners-Lee T., Cailliau R., Luotonen A. Nielsen H.F., Secret A. *The World-Wide Web*. Communications of the ACM, 37(8), 1994, pp. 76-82.

[Bea00] Bea WebLogic Portal, 2000; http://e-docs.bea.com/wlac/portals/docs/portlet.html#what

[BeGr98] Benn W., Gringer I. *Zugriff auf Datenbanken über das World Wide Web*. Informatik-Spektrum, 21(1), Springer, 1998, pp. 1-8.

[CGI01] *The CGI Specification, Version 1.1*. University of Illinois, Urbana-Champaign, http://hoohoo.ncsa.uiuc.edu/cgi/interface.html (19 August 2001).

[Chap96] Chappell, David. Understanding ActiveX and OLE. Redmond, WA: Microsoft Press, 1996. ISBN 1-572-31216-5.

[Cook02] *Persistent Client State http Cookies*, Netscape Communication Cooperation, http://www.netscape.com/newsref/std/cookie_spec.html, 1998.

[EhKR98] Ehmayer G., Kappel G., Reich S. *Connecting Databases to the Web: A Taxonomy of Gateways*. Proc. of the 8th Int. Conference on Database and Expert Systems Applications (DEXA), Springer LNCS 1308, Toulouse, France, September 1997, pp. 1-15.

[FCGI02] *FastCGI*. http://www.fastcgi.com.

[FGLM98] Fankhauser P., Gardarin G., Lopez M., Munoz J., Tomasic A., *Experiences in Federating Databases: From IRO-DB to MIRO-Web*, Proc. of the 24th Int. Conference on Very Large Data Bases (VLDB´98), A. Gupta et al. (eds.), New York City, USA, Morgan Kaufmann, Aug. 1998.

[FGMF97] Fielding, R., Gettys, J.; Mogul, J.; Frystyk, H.; Berners-Lee, T.: RFC 2068 – *Hypertext Transfer Protocol – HTTP/ 1.1*. MIT 1997.http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2068.txt

[HTTP01] *Hypertext Transfer Protocol (HTTP)*. World Wide Web Consortium (W3C), http://www.w3.org/Protocols/ (19 August 2001).

[ISAPI] Microsoft: Internet Server API Documentation.http://www.microsoft.com/win32dev/apiext/isalegal.htm

[JaSc99] *JavaScript Documentation*. Netscape Inc. 1999, http://developer.netscape.com/docs/manuals/javascript.html (19 August 2001).

[JaSer99] *Java Servlets Tutorial*. Sun Microsystems Inc. January 1999, http://developer.java.sun.com/developer/onlineTraining/Servlets/ (19 August 2001).

[JDBC99] *The JDBC Database Access API*. http://java.sun.com/products/jdbc (19 August 2001).

[JSP01] JSP, 2001. http://www.jsp-develop.de/newsletter/6/

[LiNa99 Liu, Juhne; Narayanan, Naru: Enterprise Java Developers Guide (Enterprise Computing Series); McGraw-Hill; ISBN: 0071346732; Book&Cd ro edition; March 31, 1999

[Löse98 Löser H. *Techniken für Web-basierte Datenbankanwendungen: Anforderungen, Ansätze, Architekture*., Informatik Forschung und Entwicklung 13, Springer, 1998, pp. 196-216.

[MaKH00] Meyer H., Klettke M., Heuer A. *Datenbanken im WWW – Von CGI bis JDBC und XML*. HMD Praxis der Wirtschaftsinformatik. WWW & Datenbanken. Heft 214, August 2000. dpunkt.verlag. pp. 5-22.

[Mari00] Mariucci M. *Enterprise Application Server Development Environments*. Technical Report, Universität Stuttgart, October 2000.

[NgSr96] Nguyen T., Srinivasa V. *Accessing Relational Datebases from the World Wide Web*. Proc. of SIGMOD Conference, Jagadish H.V. et al. (eds.), Montreal, Canada, 1996.

[NSAPI] Netscape: NSAPI Basics: http://developer.netscape.com/docs/manuals/enterprise/nsapi/svrop.htm

[Ora00] Oracle Portal 3.0, 2000; http://www.oracle.com/ip/ids/index.html?portal.html

[OWE] Oracle9i Wireless Edition, http://technet.oracle.com/products/iaswe/content.html

[PrRW01] Pröll B., Retschitzegger W., Wagner R.R. *TIScover – Eine generische Plattform für webbasierte Tourismusinformationssysteme*. Informatik Forschung und Entwicklung, Themenheft Electronic Commerce, Springer Verlag, 16(1), 2001, pp. 1-13.

[Ragg96] Raggett,D.:*HTML* Addison-Wesley Verlag; 3.te Auflage; 1996

[Seeg96] Seeger, J.: *MIME-Types im WWW-Umfeld*. i'X, Heise Verlag, S. 142 ff; 1/1996

[Shkl96] Shklar L. *Web Access to Legacy Data*. Proc. of the 5th International World Wide Web Conference (WWW), Paris, France, May 1996.

[SKRP01] Schrefl M., Kapsammer E., Retschitzegger W., Pröll B. *Self-Maintaining Web Pages – An Overview.* Proc. of the 12th Australasian Database Conference, Queensland, Australia, January/February, 2001.

Spielmann S. JSP versus XSP. O´Reilly Network, http://www.onjava.com/lpt/a//onjava/2001/02/22/jsp_servlets.html.

[SSJS98] *Server-Side JavaScript Guide*. Netscape Communications Corporation, http://developer.netscape.com:80/docs/manuals/js/server/jsguide/contents.htm (19 August 2001).

[Tura01] Turau, Volker Die Grundlagen Java Server Pages: Dynamische Generierung von Web-Dokumenten; 2.te Auflage 2001; dpunkt.verlag; ISBN 3-89864-131-7

[Tura99] Turau V. Te*chniken zur Realisierung Web-basierter Anwendungen*. Informatik-Spektrum 22, Springer, 1999, pp. 3-12.

[WPS] IBM Websphere Portal Server, http://www-4.ibm.com/software/webservers/portal/

[WSA] W3C Web Services Activity, http://www.w3.org/2002/ws/

[WSPR] Oasis Web Services Remote Portal (WSRP) Technical Commitee, http://www.oasis-open.org/committees/wsrp/

[WTP] IBM Websphere Transcoding Publisher, http://www-4.ibm.com/software/webservers/transcoding/

5 Speicherung von XML-Dokumenten

Meike Klettke, Holger Meyer, Werner Retschitzegger, Rainer Unland

Kurzfassung

Für viele Anwendungen ist es notwendig, XML-Dokumente zuverlässig, effizient und dauerhaft zu speichern. Entsprechend ihres Aufgabenspektrums sind Datenbanksysteme erste und ernsthafte Kandidaten für die Dokumentablage und -wiederauffindung. Allerdings unterscheiden sich das Datenmodell von XML und besonders das relationale Datenbankmodell doch erheblich. Entsprechend ist der Abbildungsprozess nicht einfach. Im Gegenteil, es gibt eine Reihe unterschiedlicher Methoden, wie Dokumente mehr oder weniger zusammenhängend, strukturerhaltend oder -ändernd, voll oder teilinterpretiert in Datenbanksystemen abgelegt und wieder aufgefunden werden können. Dieses Kapitel stellt eine Klassifikation vor, ordnet die wesentlichen Ansätze darin ein und bewertet sie anhand eines durchgängigen Beispiels.

5.1 Motivation

Mit der stark zunehmenden Verbreitung von XML gibt es auch immer mehr Anwendungen, die XML-Dokumente dauerhaft speichern möchten. In diesem Kapitel wird zunächst eine Klassifikation von XML-Dokumenten vorgenommen, um anschließend Anforderungen an deren Speicherung zu diskutieren. Es folgt eine detaillierte Darstellung der zwischenzeitlich recht zahlreichen Methoden zur Speicherung von XML-Dokumenten. Das Prinzip der einzelnen Speicherungsmethoden wird vorgestellt, die jeweiligen Methoden werden anhand von Beispielen erläutert und dabei auch einer entsprechenden Bewertung unterzogen. Es finden dafür verschiedene Kriterien Anwendung, wie die Eignung für verschiedene Arten von XML-Dokumenten, die Wiederherstellbarkeit der gespeicherten XML-Dokumente und die Effizienz von Anfragen. Eine derartige Bewertung ist insbesondere deshalb von Nutzen, da keine der vorgestellten Methoden für alle denkbaren Anwendungsbereiche gleichermaßen geeignet ist. Eine große Rolle spielt, welche Arten von Informationen in den XML-Dokumenten gespeichert sind, wie diese Informationen weiterverarbeitet werden sollen, wie die Anfragen erfolgen sollen usw.

Es sei darauf hingewiesen, dass auf Basis der in diesem Kapitel erarbeiteten Klassifikation von Speicherungsmethoden in Kapitel 13 konkrete Werkzeuge und

Systeme vorgestellt werden, die zur Speicherung von XML-Dokumenten eingesetzt werden können, und es wird dort gezeigt, wie kommerzielle Systeme derzeit vorgehen.

5.2 Klassifikation

5.2.1 Klassifikation von XML-Dokumenten

XML-Dokumente können für sehr unterschiedliche Anwendungen eingesetzt werden. Dabei können sie sowohl strukturierte Daten als auch Informationen mit Volltextcharakter beinhalten. Die Dokumenteigenschaften haben Auswirkungen auf die Verarbeitung der Dokumente. Eine Unterteilung in daten- und dokumentorientierte XML-Dokumente, die im Folgenden vorgestellt wird, ist deshalb allgemein akzeptiert. Erweitern werden wir diese Unterteilung um eine dritte Klasse, die wir als gemischt strukturierte XML-Dokumente bezeichnen werden.

Datenorientierte XML-Dokumente. Der Inhalt *datenorientierter XML-Dokumente* ist meist feingranular strukturiert und getypt, die Ordnung der Elemente ist nicht signifikant und das Schema stark anwendungsspezifisch. Solche Dokumente sind meist das Resultat einer automatischen Generierung; Anwendungen greifen häufig auf einzelne Bereiche im Dokument zu. Typische Beispiele sind etwa Produktkataloge, Bestellungen und Rechnungen.

Dokumentorientierte XML-Dokumente. XML-Dokumente werden als *dokumentorientiert* bezeichnet, falls die darin enthaltenen Daten eher grobgranular und unstrukturiert vorliegen, vorwiegend ungetypt sind, einer signifikanten Ordnung unterliegen und häufig kein explizites Schema aufweisen. Sie werden oft manuell erstellt. Die kleinste sinnvoll zu verarbeitende Einheit ist meist das Dokument selbst, das daher bei der Speicherung als Ganzes erhalten bleiben oder originalgetreu wiederhergestellt werden muss. Beispiele für dokumentorientierte XML-Dokumente sind wissenschaftliche Artikel oder Geschäftsberichte von Unternehmen.

Gemischt strukturierte XML-Dokumente. Bei XML-Dokumenten ist häufig eine eindeutige Klassifikation in datenorientiert oder dokumentorientiert nicht möglich, da Eigenschaften beider Varianten vorhanden sind. Solche Mischformen werden daher *gemischt strukturierte XML-Dokumente* genannt. Ein Beispiel dafür sind Online-Buchläden mit datenorientierten Informationen über Bücher wie Titel, Verlag, Preis und Lieferzeit sowie dokumentorientierten Inhaltsangaben und Rezensionen. Ein Beispiel aus dem Bereich Web-basierter Tourismusinformationssysteme ist die Repräsentation von Hotels mit datenorientierten Informationen wie Hotelname, Angaben zur Kategorie, Adresse und Preis sowie dokumentorientierten Inhalten wie textuelle Beschreibungen zu Haus und Ort, Veranstaltungen und Anreise.

5.2.2 Anforderungen an die Speicherung von XML-Dokumenten

Bei der Speicherung von XML-Dokumenten muss gewährleistet sein, dass Anfrage- und Änderungsoperationen korrekt und effizient den Zugriff auf XML-Dokumente oder auf Teile davon realisieren können. Der Einsatz von SQL oder OQL zur Arbeit auf XML-Dokumenten ist aus Nutzersicht nur für einen Teilbereich der Anwendungen geeignet. Deshalb sollten eher speziell auf XML zugeschnittene Anfragesprachen wie XPath und/oder XQuery genutzt werden. Auch benötigt man eine Transaktionsverwaltung. Weiterhin soll eine Unterstützung des Simple Access Protocols (SAX) und des Document Object Model (DOM) gegeben sein (vergleiche dazu auch [FiKM01]).

Datenorientierte XML-Dokumente verfügen über ein gegebenes Schema, das sich nur sehr selten oder nie verändert. Daraus ergibt sich, dass Anfragen an die Dokumente nur die Inhalte der Dokumente erfragen. Änderungen beziehen sich ebenfalls nur auf Inhalte. Als Anfragesprache eignen sich neben XPath und XQuery auch SQL und OQL. Zur Änderung der gespeicherten Informationen lassen sich DOM-Operationen verwenden, oder sie können mit SQL bzw. OQL durchgeführt werden. Bei datenorientierten XML-Dokumenten müssen die gespeicherten Informationen erhalten bleiben und wiederherstellbar sein. Es ist jedoch nicht erforderlich, dass die XML-Dokumente in genau derselben Syntax/Reihenfolge wiederhergestellt werden können.

Demgegenüber haben *dokumentorientierte XML-Dokumente* in der Regel kein Schema. Anfragen und Änderungsoperationen beziehen sich auf Struktur und Inhalt. Als Anfragesprache kommen Xpath, XQuery oder Anfragesprachen aus dem Information Retrieval in Frage. Änderungen können über DOM-Methoden vorgenommen werden. Es muss eine vollständige Wiederherstellbarkeit der XML-Dokumente (inkl. der ursprünglichen Anordnung der Elemente) gewährleistet sein.

Gemischt strukturierte XML-Dokumente weisen ein optionales oder ein partiell vorhandenes Schema auf. Anfragen und Änderungen können sich auf Struktur und Inhalt beziehen. XPath und XQuery sind wiederum sehr gute Kandidaten für Anfragen, während DOM-Methoden für Änderungen eingesetzt werden können. Auch hier ist eine passgenaue Wiederherstellbarkeit der XML-Dokumente erforderlich.

5.2.3 Klassifikation der Speicherungsvarianten

Mit der allgemeinen Akzeptanz von XML als Format zur Darstellung semistrukturierter Daten wurden auch Ansätze zur Speicherung von XML-Dokumenten entwickelt. Mittlerweile ist akzeptiert, dass *alternative Ansätze* notwendig sind, da XML-Dokumente so unterschiedlich sein können, dass es keine für alle Anwendungen »beste Variante« gibt.

1. Speicherung als Ganzes. XML-Dokumente können als *Ganzes* in *Datei-* oder *Datenbanksystemen (DBS)* gespeichert werden. In Letzteren können sie z.B. als

Attribute vom Typ BLOB (Binary Large Object), CLOB (Character Large Object) oder mittlerweile meist auch vom Typ XML abgelegt werden. Für die Realisierung von effizienten Zugriffen können dabei *Indexe* über den Dokumentkollektionen angelegt werden[1]. Indexiert werden können bei den Verfahren sowohl die Inhalte der Dokumente (Volltextindex) als auch die Struktur (Struktur- oder Pfadindex). Anfragen werden dann unter Auswertung der Indexe beantwortet. Die Indexierung von XML-Strukturen wird in Abschnitt 5.4 kurz dargestellt, in Kapitel 8 werden XML-Indexstrukturen im Detail vorgestellt, die Volltextindexierung wird in Kapitel 9 erläutert. Die Speicherung von XML-Dokumenten als Dateien und die Indexierung über diesen Dateien ist für dokumentorientierte Anwendungen geeignet.

2. Dekomposition und generische Speicherung. Die zweite Klasse von Speicherungsmethoden nimmt im Unterschied zur ersten Klasse eine *Dekomposition der XML-Dokumente* und eine Speicherung auf Basis *generischer Modelle* vor[2]. Generisch bedeutet in diesem Zusammenhang, dass die XML-Dokumente unabhängig von der jeweiligen Anwendung und dem Vorhandensein einer DTD oder eines Schemas transformiert werden. Die Speicherung folgt dabei typischerweise auf Basis eines Graphmodells, z.B. den Informationen des Document Object Model folgend. Diese Vorgehensweise eignet sich insbesondere bei gemischt strukturierten Dokumenten. Die Herangehensweise wird in Abschnitt 5.5 näher vorgestellt. Bei dieser Speicherungsform ist die Verwendung eines Datenbanksystems optional.

3. Strukturierte Speicherung in Datenbanken. Es gibt Verfahren, die auf der Basis von Datenbanksystemen eine strukturierte Speicherung von XML-Dokumenten realisieren. Im Gegensatz zur generischen Speicherung erfolgt hier eine anwendungsbezogene, d.h. auf die spezifische Dokumentstruktur abgestimmte, Abbildung (Mapping) auf ein Datenbankschema, wobei eine Speicherung mit Hilfe von *relationalen* oder *objektrelationalen* bzw. *objektorientierten* DBS erfolgen kann. Der Einsatz von DBS und die Abbildung der Dokumentstruktur auf die Datenbankstruktur ist vor allem für datenorientierte Anwendungen sinnvoll. Eine genaue Darstellung dieser Speicherungsmethode erfolgt in Abschnitt 5.6.

Als weitere Kombination von Speicherungsart und verwendetem System kann eine *hybride Speicherung* identifiziert werden. Dabei werden XML-Dokumente aufgrund anwendungsabhängiger Kriterien *aufgeteilt*, wobei *verschiedene Speicherungsmethoden* für die einzelnen Teile Verwendung finden können. Zum Beispiel können bei Informationen über Bücher die Metadaten, die das Bibliothekswesen verwendet, in einem DBS abgelegt werden, während die elektronischen Dokumente in Dateien gespeichert werden, und dabei wiederum jeweils zerlegt

1. Das Verfahren wird auch als *textbasierte »native« XML-Speicherung* bezeichnet. Eine Erklärung ist in [Bour02] zu finden.
2. Das Verfahren wird auch *modellbasierte »native« XML-Speicherung* genannt. Eine Diskussion des Begriffes ist in [Bour02] zu finden.

oder als Ganzes. Bei dieser Methode, die in Abschnitt 5.7 näher vorgestellt wird, wird zur Speicherung von XML-Dokumenten eine Kombination der vorgestellten Methoden eingesetzt.

Insgesamt lassen sich bei den einzelnen Speicherungsmethoden folgende Merkmale unterscheiden:

- Speicherung der XML-Dokumente erfolgt als Ganzes oder zerlegt (fragmentiert).
- DTD bzw. Schema zur Speicherung von XML-Dokumenten ist erforderlich bzw. nicht erforderlich.
- Vollständige Wiederherstellbarkeit der XML-Dokumente ist gegeben bzw. nicht gegeben.
- Verwendung eines generischen XML-Modells (z.B. Graphenmodell, XPath, XML Infoset, DOM) oder nicht.
- Speicherung erfolgt auf Basis eines (relationalen, objektrelationalen oder objektorientierten) DBS vs. Speicherung erfolgt ohne DBS.
- Benutzerdefinierte Beschreibung der Abbildung vs. automatische (nicht beeinflussbare) Abbildung.

Die genannten Speicherungsmethoden werden im weiteren Verlauf des Kapitels detailliert dargestellt. Zu den jeweiligen Speicherungsmethoden werden die oben angegebenen Eigenschaften diskutiert, die die entsprechenden Methoden auszeichnen.

5.3 Fortlaufendes Beispiel

Die in den folgenden Abschnitten vorgestellten Speicherungsmethoden werden anhand eines fortlaufenden Beispiels näher erläutert. Der verwendete Realitätsausschnitt stammt aus dem Bereich Web-basierter Tourismusinformationssysteme [PrRW01]. Abb. 5-1 zeigt eine DTD, die Hotels hinsichtlich Kategorie und Lokation näher charakterisiert.

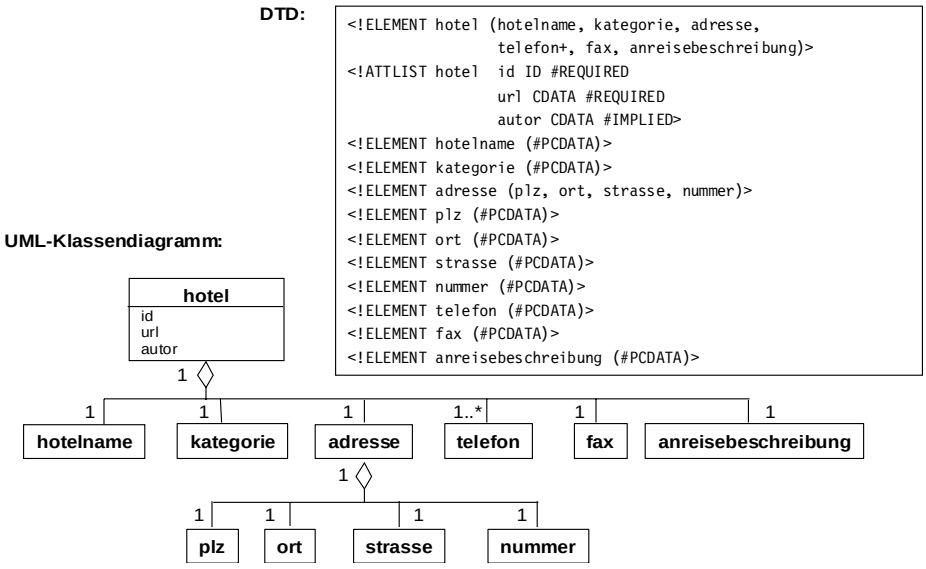


Abb. 5-1: *DTD des Hotelbeispiels*

Aus Gründen der besseren Lesbarkeit ist diese DTD auch als UML-Klassendiagramm dargestellt, wobei Elemente als Klassen, geschachtelte Elemente als Aggregationen und XML-Attribute als Attribute von Klassen repräsentiert werden [Carl01]. Abb. 5-2 zeigt ein entsprechendes gültiges XML-Dokument.

Die Werte der Elemente (`#PCDATA`) werden dabei in eigenen Attributen (`wert`) direkt bei den entsprechenden anonymen Elementobjekten gespeichert.

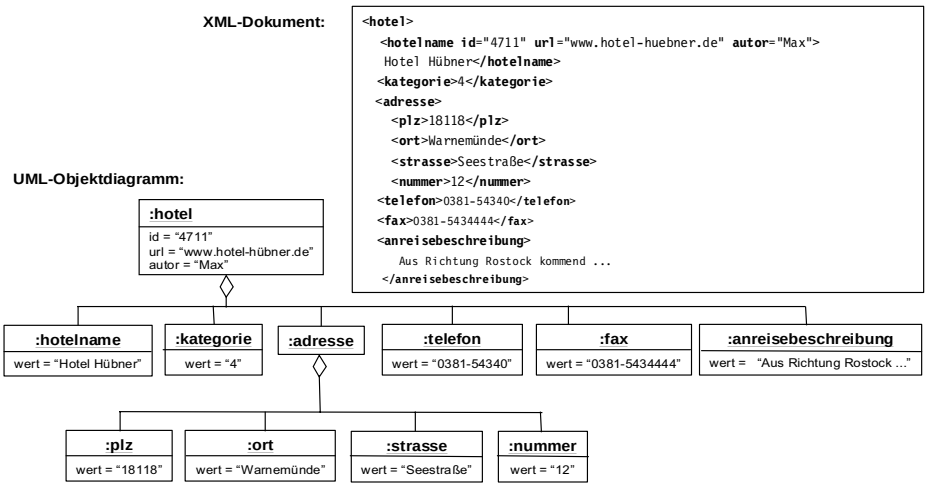


Abb. 5-2: *XML-Dokument eines Hotels*

5.4 Speicherung von XML-Dokumenten als Ganzes

Für viele Anwendungen ist es sinnvoll, XML-Dokumente unverändert zu speichern und so zu indexieren, dass Anfragen an sie effizient durch Auswertung der Indexinformationen beantwortet werden können. Für die Indexierung gibt es verschiedene Möglichkeiten, die, wie erwähnt, in Kapitel 8 vorgestellt werden. Wir deuten hier der Vollständigkeit halber eine Möglichkeit an. Die einfachste Variante eines Indexes über XML-Dokumenten ist ein Volltextindex in Form einer invertierten Liste, wie er aus dem Information Retrieval bekannt ist. Eine invertierte Liste ist ähnlich dem Index eines Buches, welcher zu den wesentlichen Begriffen jeweils eine Seitenzahl im Buchtext angibt. Auf dem Volltextindex sind Zugriffe wie *boolesches Retrieval* möglich [SaGi83]. Abb. 5-3 zeigt den prinzipiellen Aufbau einer solchen Liste.

Einige Beispielanfragen sind:

- Hotel `AND` Warnemünde
- Hotel `AND` Strand
- (Hotel `OR` Pension) `AND` (Warnemünde `OR` Rostock)

Beantwortet werden diese Anfragen, indem zu den Stichwörtern die Menge der Dokumente herausgesucht wird, in der die Stichwörter auftreten. Bei Anfragen mit `AND`-Verknüpfung wird untersucht, welches Dokument bei allen Stichwörtern gespeichert ist. Anfragen mit `OR`-Verknüpfungen erwarten, dass lediglich ein Stichwort zutrifft. Die Negation von Anfragebestandteilen mit `NOT` ist ebenfalls unter Zuhilfenahme des Indexes beantwortbar.

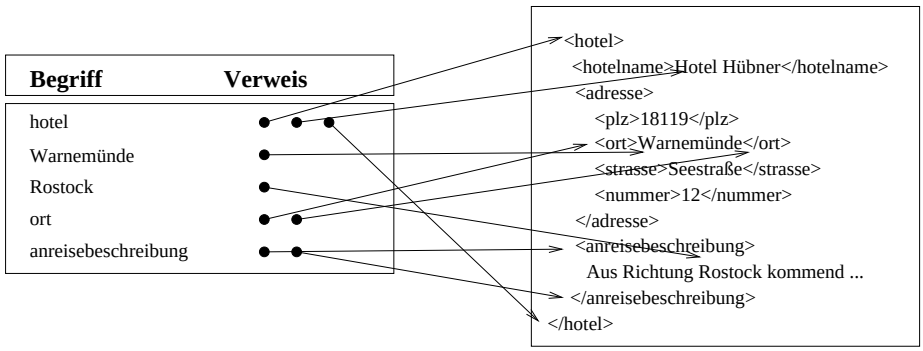


Abb. 5-3: *Indexspeicherung in Form einer invertierten Liste*

Die Speicherung der Position der Stichwörter im Dokument erlaubt eine Form *kontextbezogener Anfragen* wie beispielsweise: Finde alle Dokumente, bei denen

- Suchbegriff A vor B im Dokument auftritt oder
- der Begriff B unmittelbar auf A folgt oder
- A und B mit einem Abstand von höchstens 5 Wörtern auftreten.

Nicht realisierbar sind hingegen Anfragen, welche die *Kenntnis der Struktur von XML-Dokumenten* erfordern. Sollen im Index auch solche Informationen ausgewertet werden, muss dieser wie in Abb. 5-4 schematisch dargestellt erweitert werden. Zum Stichwort-Index, der Stichwörter und Verweise auf Dokumente enthält, wird hier zusätzlich gespeichert, in welchem Element der Wert auftritt. Elementinformationen werden separat gespeichert. Die Elementhierarchie wird durch den Eintrag `Vorgänger`, der wiederum auf diese Tabelle verweist, ausgedrückt.

Man sieht, dass die Indexinformationen in diesem Fall umfangreicher und komplexer strukturiert sind. Durch einen auf diese Weise erweiterten Index lassen sich jetzt Anfragen realisieren, die die XML-Struktur nutzen, z.B.

- hotel.ort `CONTAINS` (»Warnemünde«)
- hotel.ort `CONTAINS` (»Warnemünde«) `AND`
  hotel.freizeitmoeglichkeiten `CONTAINS` (»Swimming Pool«)

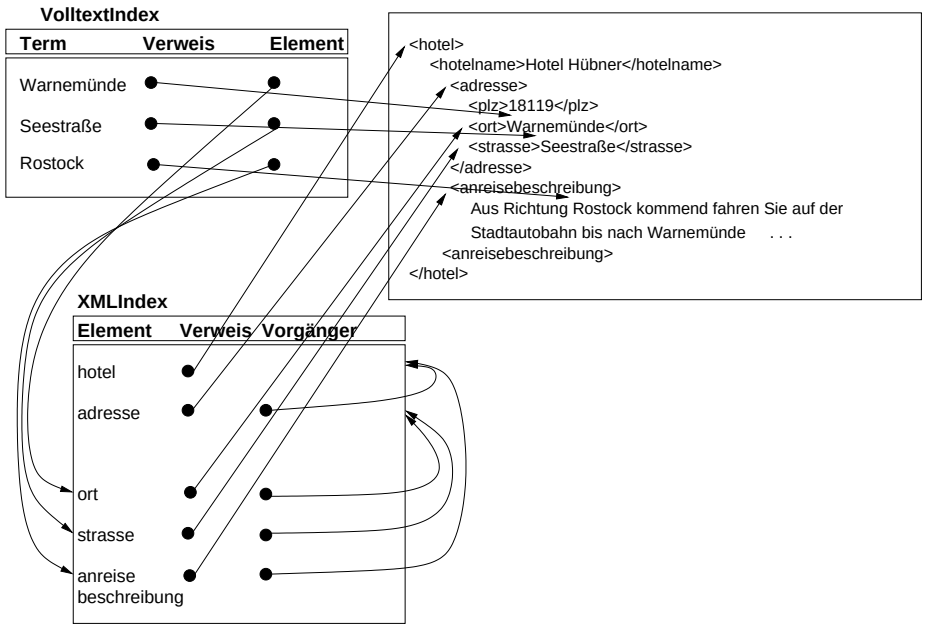


Abb. 5-4: *Indexspeicherung in Form einer invertierten Liste*

5.5 Dekomposition und generische Speicherung

Das grundlegende Merkmal einer Speicherung auf Instanzebene liegt darin, dass im Gegensatz zur Speicherung von XML-Dokumenten als Ganzes eine *Dekomposition* der Dokumentstruktur sowie dessen *Transformation in ein generisches Modell* vorgenommen wird, beispielsweise um es in einem *DBS* zu speichern. Die Speicherung erfolgt auf *Instanzebene*, d.h., die Dokumentstruktur wird zerlegt

als Werte, z.B. in Datenbankattributen, gespeichert. Beispiele für solche generischen Modelle sind das Graphenmodell, das XPath-Datenmodell, das XML Infoset oder das Document Object Model. Die Darstellung dieser Modelle findet sich in zahlreichen XML-Büchern wie [Brad98] und [HaMe01] sowie auf den Webseiten des W3C.

Im Folgenden werden zwei verschiedene, auf dem Einsatz von relationalen DBS basierende Varianten dieser Speicherungsmethode näher vorgestellt. Zunächst wird das grundlegende Prinzip der Verwendung von Graphstrukturen zur Speicherung erläutert. Danach wird eine Methode zur Verwendung von DOM zur Speicherung vorgestellt.

5.5.1 Verwendung von Graphstrukturen

Bei dieser Methode wird die Graphstruktur, die den XML-Dokumenten zugrunde liegt, zur Speicherung verwendet. XML-Dokumente werden als geordnete Graphen interpretiert, deren Knoten *Elemente* oder *Attribute* darstellen. Die Kanten repräsentieren die Beziehungen der Elemente zu den Subelementen oder Attributen [FlKo99]. Eine Speicherung als Graphstruktur erfolgt oftmals auf Basis von relationalen DBS, was allerdings nicht zwingend erforderlich ist. Wird ein DBS zugrunde gelegt, so ist das Schema dieser Datenbank (DB) darauf ausgelegt, beliebige Graphstrukturen aufzunehmen. Es ist daher vollkommen unabhängig vom Schema der XML-Dokumente.

DB-Schema. Abb. 5-5 und Tab. 5-6 zeigen zwei Relationen, die zur Verwaltung von Elementen und Attributen dienen. In der *Element*-Relation werden neben einem Elementidentifikator `ID`, Elementnamen (`elementname`) und Werte (`wert`) gespeichert. Die Abbildung der *Elementhierarchie* erfolgt über die Speicherung der `ID` des Vorgängerknotens (`vorgänger`). Das Attribut `position` wird verwendet, um eine *Ordnung* zwischen den Elementen auf der gleichen Ebene einzuführen. Zur Speicherung mehrerer XML-Dokumente wird eine eindeutige Identifikation der Dokumente (`docID`) benötigt.

Elemente

docID	elementname	ID	vorgänger	position	wert
h0001	hotel	001		1	
h0001	hotelname	002	001	1	Hotel Hübner
h0001	kategorie	003	001	2	4
h0001	adresse	004	001	3	
h0001	plz	005	004	1	18119
h0001	ort	006	004	2	Warnemünde
h0001	strasse	007	004	3	Seestraße
h0001	nummer	008	004	4	12
h0001	telefon	009	001	4	0381-54340
h0001	fax	010	001	5	0381-5434444
h0001	anreisebeschreibung	011	001	6	Aus Richtung Rostock ...

Abb. 5-5: *DB-Schema zur Speicherung der Elemente*

Attribute

docID	attributname	elementID	wert
h0001	id	001	4711
h001	url	001	www.hotel-huebner.de
h001	autor	001	Max

Abb. 5-6: *DB-Schema zur Speicherung der Attribute ([FlKo99], [ShYU99])*

Beispiel. Man sieht in Abb. 5-5, dass das Wurzelelement `hotel` keinen Verweis auf einen `Vorgänger`-Knoten hat. Beim Element `hotelname` ist ein Verweis auf die `ID` des Vorgängerknotens `hotel` gesetzt usw. Im Hinblick auf die Reihenfolge der Subelemente von `adresse` hat `plz` die `Position` 1, da diese als erstes Element auftritt, `ort` hat die `Position` 2 usw.

Die *Attribute* des Hotelbeispiels `id`, `url` und `autor` werden in einer ähnlichen Relation gespeichert. Hier ist neben der Speicherung von Attributnamen und Wert die Zuordnung zur `elementID` erforderlich. Im Beispiel haben alle Attribute die gleiche `elementID`, da alle Attribute im XML-Dokument dem Element `hotel` zugeordnet sind.

Abgrenzung zu Indexen. Der gravierendste Unterschied zu einem Index der oben erwähnten Art besteht darin, dass bei einem Index das XML-Dokument bei der Speicherung unverändert bleibt, die Indexe also nur Metainformationen darstellen. Im Gegensatz dazu wird bei der graphbasierten Speicherung das XML-Dokument selbst transformiert und anschließend gespeichert. Darüber hinaus müssen zusätzliche Informationen wie die Ordnung von Elementen verwaltet werden, um eine verlustfreie Wiederherstellbarkeit zu gewährleisten.

Anfragen. Es ist bei diesem Speicherungsverfahren möglich, XPath- oder XQuery-Anfragen zu formulieren und aus den gespeicherten Informationen die Ergebnisse zu den Anfragen abzuleiten. Dabei muss eine Transformation der

XML-Anfragen in Datenbankanfragen erfolgen sowie eine Transformation der Datenbankergebnisse in XML-Fragmente, die als Ergebnisse der Anfrage geliefert werden sollen.

Man kann auch Anfragen direkt an die Speicherungsstruktur stellen. Dazu muss diese aber bekannt sein. Im Falle der Verwendung relationaler DBMS lässt sich also SQL einsetzen, wobei die Anfragen dadurch, dass die Strukturinformationen als Werte der DB gespeichert sind, weder leicht zu formulieren sind, noch die Anfrageoptimierung der DBMS auf diese Speicherungsform zugeschnitten ist.

Verwendung der Methode. Grundsätzlich ermöglicht die Methode eine einfache Speicherung beliebiger XML-Dokumente. Zumeist werden relationale DBS als Basis für die Speicherung verwendet. Das Verfahren ist besonders für *dokumentorientierte* und *gemischt strukturierte XML-Dokumente*, also für Dokumente ohne Schema oder mit unregelmäßigem Schema geeignet. Die gespeicherten XML-Dokumente sind wiederherstellbar, die Wiederherstellung der XML-Dokumente ist jedoch sehr aufwändig, weil die Speicherung der Informationen stark fragmentiert erfolgt und die Zusammensetzung der Originaldokumente eine sehr große Anzahl von Joins über den in Relationen gespeicherten Daten erfordert. Es wurden Methoden entwickelt, die XPath-Anfragen an die so gespeicherten XML-Dokumente stellen. Dazu erfolgt eine Umsetzung auf SQL-Anfragen, die die Daten in der speziellen Speicherungsstruktur erfragen. Bei Anfragen, die sehr viele Elemente oder Attribute enthalten, ist die Umsetzung nicht effizient.

Die graphbasierte Speicherung von XML-Dokumenten in relationalen DBMS wird von Bradley [Brad98], Florescu und Kossmann [FlKo99] und Shimura, Yoshikawa und Uemura in [ShYU99] und [YASU01] beschrieben. Dabei wird von Bradley das grundlegende Prinzip dargestellt. Florescu/Kossmann wie auch Shimura, Yoshikawa und Uemura betrachten darüber hinaus die Effizienz der Methode bei Anfragen und stellen verschiedene Varianten vor. Werte können dabei gemeinsam mit den entsprechenden Elementen/Attributen gespeichert werden oder in Form von separaten Werte-Relationen. Schließlich könnte auch eine universelle Relation zum Einsatz kommen, die separate Spalten für jeden Wert aus den Spalten Elementname und Attributname beinhaltet. Eine etwas andere Form der Speicherung von XML-Dokumenten und eine damit verbundene Indexierung von Graphstrukturen wurde in [ScCo01] vorgestellt. Dort werden die Elementwerte als Text verbunden und gespeichert sowie ein Pfadindex angelegt, der Verweise auf die gespeicherten Elementwerte enthält.

5.5.2 Einsatz des Document Object Model

Zur Speicherung von XML-Dokumenten auf Instanzebene wird oftmals auch das Document Object Model verwendet, das unmittelbar auf dem im vorangegangenen Abschnitt vorgestellten Prinzip der Verwendung von Graphstrukturen basiert. DOM selbst stellt ein programmiersprachen- und anwendungsunabhängiges Objektmodell zur Repräsentation von XML-Dokumenten im Hauptspei-

cher und ein entsprechendes API zu deren Verarbeitung zur Verfügung. XML-Parser transformieren die Bestandteile von XML-Dokumenten (Elemente, Attribute, Werte, Entities etc.) in eine dem DOM-Modell entsprechende Objekthierarchie. Manche Ansätze speichern diese Hierarchien in objektorientierten DBMS, andere verwenden relationale DBMS oder auch das *Lightweight Directory Access Protocol* (LDAP) [LaMa01]. Weitere Implementierungen entwickeln eigene Speicherformate.

DOM-Klassen. Abb. 5-7 stellt die Klassenhierarchie von DOM als UML-Klassendiagramm dar.

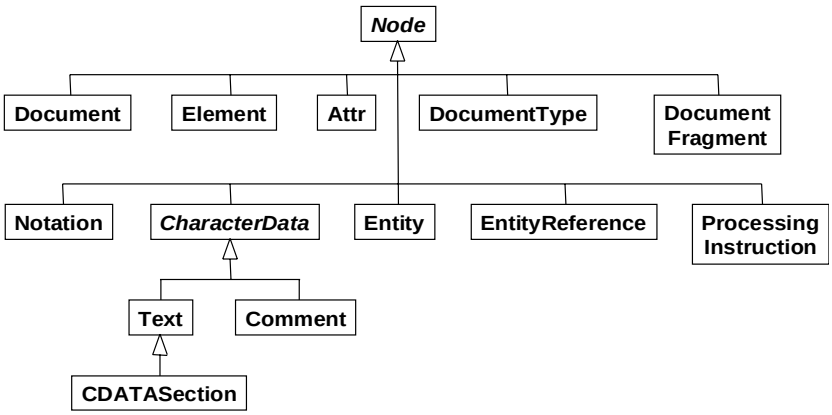


Abb. 5-7: *DOM Klassenhierarchie*

An dieser Stelle sollen nur jene DOM-Klassen kurz vorgestellt werden, deren Kenntnis zum Verständnis dieser Speicherungsmethode notwendig ist. Eine ausführliche Darstellung des Document Object Model erfolgte bereits in Kapitel 1.

Die Wurzel der DOM-Klassenhierarchie wird durch die Klasse `Node` repräsentiert. Diese Klasse ist für die Repräsentation der Graphstruktur des XML-Dokumentes zuständig und stellt eine Reihe von Methoden zur Navigation und Manipulation der Graphstruktur zur Verfügung (z.B. `getChildren()`, `getNextSibling()`, `removeChild(Node)`). Die Subklassen `Element`, `Attribut` und `Text` sind für die Verwaltung von Elementen, Attributen und Werten vorgesehen und enthalten zusätzliche Methoden, um deren spezielle Eigenschaften zu erfragen und zu manipulieren (z.B. `getAttributes()`, `setAttribute (Attribute)`, `setValue (NodeList)`).

DB-Schema. In einem (objekt)relationalen DBS kann die Speicherung der DOM-Klassen beispielsweise durch die in Abb. 5-8 dargestellten Relationen erfolgen.

Node

nodeName	nodeType	documentID	parent-Node	Previous Sibling	Next Sibling	First Child
001	Element	h0001				002
002	Element	h0001	001		003	
003	Element	h0001	001	002	004	
004	Element	h0001	001	003	009	005
005	Element	h0001	004		006	
006	Element	h0001	004	005	007	
007	Element	h0001	004	006	008	
...						
011	Element	h0001	001	010		
012	Attribut	h0001	001		013	
013	Attribut	h0001	001	012	014	
014	Attribut	h0001	001	013		

Element

nodeName	tagName	text
001	hotel	
002	hotelname	Hotel Hübner
003	kategorie	4
004	adresse	
005	plz	18119
006	ort	Warnemünde
007	strasse	Strandstraße
...		
011	anreisebeschreibung	Aus Richtung Rostock ...

Attribut

nodeName	attributName	attributeValue	specified
012	id	4711	true
013	url	www.hotel-huebner.de	true
014	autor	Max	true

Abb. 5-8: *Speicherung der Informationen der Klassen* Node, Element, Attribut und Text

Die Attribute der Relationen basieren auf den Methoden der DOM-Klassen wie in [DOM02] definiert. Bei Einsatz von relationalen DBMS werden zusätzlich Datenbankattribute zur Repräsentation der Beziehungen zwischen den Relatio-

nen eingeführt. Bei objektorientierten und objektrelationalen DBMS werden die Verbindungen durch Vererbungshierarchien dargestellt.

Anfragen. Es ist bei dieser Speicherungsmethode möglich, eine *XML-Anfragesprache* zu verwenden. Die so erstellten Anfragen werden dann – im Falle einer Speicherung in einem DBS – durch ein Query-Rewriting-Verfahren auf Datenbankanfragen abgebildet, und die Anfrageergebnisse werden entsprechend zurücktransformiert. Wird – im Falle von DOM – kein DBS zur Speicherung genutzt, so werden die XML-Anfragen über Query-Rewriting auf DOM-Zugriffsmethoden abgebildet. Ein Vorteil liegt schließlich darin, dass über DOM nicht nur Anfragen, sondern auch Updates auf den Dokumenten vorgenommen werden können. Das stellt einen besonderen Vorteil dar, da es gegenwärtig noch keine XML-Update-Sprache gibt, die alternativ dazu verwendet werden könnte.

Ein wesentlicher Nachteil bei der Verwendung von DBS zur Speicherung besteht im Hinblick auf *Datenbankanfragen* darin, dass deren Komplexität durch das feststehende Schema und die Speicherung der Strukturinformationen auf Werteebene steigt und damit auch eine entsprechende Optimierung erschwert wird. Man sucht zum Beispiel nicht nach allen Hotels, sondern würde für eine solche Anfrage die Werte aller Elemente, deren Tag-Name `hotel` ist, ermitteln. Im Hinblick auf die Verwendung von Entities müssen die entsprechenden Referenzen aus Gründen der Anfrageeffizienz in jedem Fall bereits vor der Speicherung aufgelöst werden.

Neben dem Einsatz von XML-Anfragesprachen ist es auch hier prinzipiell möglich, die XML-Informationen direkt in dem System anzufragen, das zur Speicherung verwendet wird. Im Falle einer Speicherung in objektorientierten DBMS lässt sich also OQL als Anfragesprache verwenden. Im Falle relationaler DBS kann SQL eingesetzt werden, sofern die Struktur des Datenbankschemas jeweils bekannt ist.

Verwendung der Methode. Grundsätzlich ermöglicht die Methode eine einfache Speicherung beliebiger XML-Dokumente, ist jedoch besonders für *dokumentorientierte* und *gemischt strukturierte XML-Dokumente* geeignet. Gespeicherte XML-Dokumente sind ohne Informationsverlust wiederherstellbar, wenn dieser Prozess auch sehr aufwändig ist. Ein besonderer Vorteil von DOM liegt darin, dass es schon seit geraumer Zeit als W3C-Recommendation vorliegt und daher allgemein akzeptiert ist.

Es existieren in DOM Methoden zur Abfrage der Informationen und zur Manipulation von XML-Dokumenten. Die Verwendung von XML-Anfragesprachen ist ohne Einschränkungen möglich.

Ein Verfahren, das die Informationen aus dem DOM auf der Basis relationaler DBMS darstellt, wurde von Edwards und Hope in [EdHo00] vorgestellt. Die Verwendung der Informationen aus dem DOM zur Speicherung von XML-Dokumenten mit dem *Lightweight Directory Access Protocol* (LDAP) ist Inhalt des Artikels von Lausen und Marron [LaMa01]. Es stellt eine Möglichkeit zum effizienten Speichern von Informationen dar und ist besonders geeignet, um hierar-

chische Strukturen abzubilden. LDAP wurde als offener Standard für globale oder lokale Verzeichnisdienste im Netzwerk und im Internet entwickelt. Es kann beliebige Informationen verarbeiten und ist so konzipiert, dass ein hohes Anfragevolumen unterstützt wird, wobei sich die Daten jedoch nicht sehr häufig ändern sollten. Zahlreiche XML-DBMS setzen das DOM als Basis für die Speicherung von XML-Dokumenten ein. Einige dieser Systeme werden in Kapitel 13 vorgestellt.

5.6 Strukturierte Speicherung in Datenbanken

Während bei der im letzten Abschnitt vorgestellten Methode die Speicherung der Struktur von XML-Dokumenten auf Instanzebene erfolgt, wird nun eine Methode vorgestellt, bei der die *XML-Dokumentstruktur auf Schemaebene* repräsentiert wird. Eine notwendige Voraussetzung dafür ist ein explizites Schema der XML-Dokumente, das bei der Speicherung auf *ein DB-Schema* abgebildet wird. Damit ist das DB-Schema nicht feststehend wie bei einer generischen Speicherung auf Instanzebene, sondern vielmehr anwendungsspezifisch. Voraussetzung für diese Speicherungsmethode ist die Existenz einer *expliziten Repräsentation der XML-Dokumentstruktur* in Form eines XML-Schemas oder einer DTD.

Aufgrund der hierarchischen Schachtelungsmöglichkeiten von XML-Dokumenten bieten sich objektorientierte DBMS und objektrelationale DBMS für diese Speicherungsmethode an. Es kommen jedoch auch häufig relationale DBS zum Einsatz. In Abschnitt 5.6.1 erläutern wir eine Abbildung auf objektrelationale DBMS. In Abschnitt 5.6.2 wird gezeigt, welche zusätzlichen Schritte für die Abbildung auf relationale DBMS erforderlich sind. Ist zu einer Menge von XML-Dokumenten auch ein XML-Schema verfügbar, ergeben sich weitere Möglichkeiten der adäquaten Abbildung auf objektrelationale DBS, worauf hier jedoch nicht näher eingegangen werden kann.

Abbildungsvorschriften. Unabhängig von der Art des verwendeten DBMS müssen entsprechende *Abbildungsvorschriften* festgelegt werden, die angeben, welche XML-Dokumentstrukturen auf welche Teile des DB-Schemas abgebildet werden.[3] Abbildungsvorschriften können *vom System per Default vorgegeben* werden, können aber auch *benutzerdefinierbar* sein. Sie dienen im Wesentlichen dazu, bei einer Anfrage eine entsprechende *Transformation der Daten* vornehmen zu können, werden aber auch eingesetzt, um das *DB-Schema* aus XML-Dokumentstrukturen heraus zu *generieren*, falls das Schema z.B. in Form einer Legacy-DB nicht bereits vorhanden ist. Im Hinblick auf die Repräsentation der Abbildungsvorschriften kann unterschieden werden, inwieweit diese in der

3. Es sei an dieser Stelle darauf hingewiesen, dass auch bei der Speicherung auf Instanzebene, die in Abschnitt 5.5 vorgestellt wurde, Abbildungsvorschriften notwendig sind. Diese sind jedoch insbesondere aufgrund des feststehenden DB-Schemas und der Speicherung aller Bestandteile von XML-Dokumenten einfach zu realisieren.

Anwendung *fix-verdrahtet* sind oder aber *separat* von dieser vorliegen. Zunächst werden in den Abschnitten 5.6.1 und 5.6.2 automatische Speicherungsverfahren vorgestellt. Während bei den Diskussionen in den vorherigen Abschnitten immer die DTD als Beschreibung der Dokumentstruktur impliziert wird, folgen in Abschnitt 5.6.3 Verfahren, die eine benutzerdefinierte Abbildung verwenden.

5.6.1 Abbildung auf objektrelationale Datenbanksysteme

Die prinzipiellen Regeln für die Abbildung von XML-Dokumenten auf ein objektrelationales DB-Schema zeigt Tab. 5-1, wobei unter objektrelational das durch SQL:1999 vorgegebene Datenmodell verstanden werden soll, allerdings mit einer entscheidenden Ausnahme: `SET` und `LIST`, die in einigen DBS wie z.B. Informix bereits unterstützt werden, werden als durch das objektrelationale Datenmodell unterstützte Typkonstruktoren angesehen.

Ausgangspunkt der Betrachtung sind hier DTDs. Elemente einer DTD werden auf Datenbankattribute abgebildet. Durch die Quantifizierer wird bestimmt, ob Nullwerte erlaubt sind oder ob Listen oder Mengen von Attributen zur Speicherung verwendet werden. `LIST` ist immer dann zu verwenden, wenn entweder die Ordnung eine Rolle spielt oder dasselbe Element mehrfach vorkommen kann. `SET` hingegen repräsentiert eine ungeordnete Menge ohne Duplikate. Geschachtelte Elemente werden als `ROW`-Typen abgebildet. XML-Attribute werden ebenfalls auf Datenbankattribute abgebildet. Die Angabe `IMPLIED` oder `REQUIRED` beeinflusst, ob Nullwerte in dem Datenbankattribut auftreten können oder nicht.

XML-Information		Datenbankinformation
Elemente	XML-Element	Attribut einer Relation
	Sequenz von Elementen	Attribute einer Relation
	Alternative von Elementen	Attribute einer Relation, Nullwert jeweils erlaubt oder neue Relation(en)
	Element mit Quantifizierer ?	Attribut einer Relation, Nullwert erlaubt
	Elemente mit Quantifizierer (+, *)	`SET` oder `LIST` oder eine neue Relation
	Komplex strukturiertes (geschachteltes) Element	`ROW`-Typ
	Geschachteltes Element mit Quantifizierer + oder *	`LIST` (`ROW`-Typ)
Attribute	XML-Attribut	Attribut einer Relation
	IMPLIED/REQUIRED	Nullwert erlaubt/nicht erlaubt
	Defaultwert	Default

Tab. 5-1: *Abbildung von XML auf objektrelationale DBS*

Objektrelationales DB-Schema für das Hotelbeispiel. Gemäß diesen Abbildungsvorschriften hat die DB für das Hotelbeispiel das in Abb. 5-9 dargestellte Aussehen:

Hotel

id	url	autor	hotelname	kategorie	adresse				...
					plz	ort	strasse	nummer	
0001	www.hotel-huebner.de	Max	Hotel Hübner	4	18119	Warne-münde	Seestrasse	12	...

Abb. 5-9: *Objektrelationales DB-Schema für das Hotelbeispiel*

Das Beispiel lässt sich mit einem objektrelationalen DBS adäquat umsetzen. Das Beispiel in Abb. 5-10 zeigt dazu exemplarisch die Syntax, wie sie vom DBMS Informix verwendet wird. Man vergleiche hierzu auch Kapitel 13.

```
CREATE TABLE      hotel
      (id         VARCHAR(19)                        NOT NULL,
       url        VARCHAR(60)                        NOT NULL,
       autor      VARCHAR(30),
       hotelname  VARCHAR(50)                        NOT NULL,
       kategorie  INTEGER                            NOT NULL,
       adresse    ROW
                    (plz      INTEGER                NOT NULL,
                     ort      VARCHAR(50)            NOT NULL,
                     strasse  VARCHAR(30)            NOT NULL,
                     nummer   INTEGER                NOT NULL
                    ) NOT NULL,
       telefon    LIST (VARCHAR(30)                  NOT NULL,
       fax        VARCHAR(30)
      ),
```

Abb. 5-10: *Beispiel: Erzeugung des Datenbankschemas*

Probleme mit unbekannten Datentypen. Im obigen Beispiel wurden Datentypen für die Datenbankattribute festgelegt. Während die Bezeichnungen der Datenbankattribute sowie die Anordnung und Verschachtelung aus der DTD abgeleitet werden können, sind die Informationen über die Wertebereiche nicht verfügbar. Diese müssen entweder vom Anwender explizit angegeben werden, oder es wird anstelle einer DTD ein XML-Schema als Ausgangspunkt für den Abbildungsprozess eingesetzt. Dieser zweite Aspekt wird hier nicht im Einzelnen behandelt; wir verweisen dazu auf Kapitel 2.

Problem mit Bezeichnern. Man sieht an dem Beispiel, dass alle Teile eines XML-Dokumentes strukturiert dargestellt sind, indem alle Elemente und Attribute auf DB-Attribute abgebildet wurden. Werden die Bezeichner der Elemente und Attri-

bute als Bezeichner für DB-Attribute übernommen, so kann eventuell eine entsprechende Anpassung erforderlich sein, da der XML-Datentyp `NMTOKEN`, dem XML-Bezeichner unterliegen, gewisse Sonderzeichen erlaubt, die in DBS nicht erlaubt sind. Eine Standardisierung der Abbildung von Bezeichnern eines XML-Dokumentes auf Relationen- und Attributnamen von DBS wird in [SQLXML02] festgelegt.

Problem mit Rekursionen. Ein Fall, der sich mit diesen Abbildungsvorschriften nicht so einfach realisieren lässt, sind *Rekursionen*. Während in XML-Dokumenten keine Rekursionen auftreten können, da die Dokumente endlich sind, ist dies in DTDs sehr wohl möglich. Das folgende Beispiel soll den Fall erläutern:

```
<!ELEMENT ort (name, lage, einwohnerzahl, ausflugsziel*)>
        ...
<!ELEMENT ausflugsziel (ort, entfernung, beschreibung)>
```

Die Beschreibung von Orten (`ort`) enthält dabei neben anderen Informationen auch Vorschläge für Ausflüge (`ausflugsziel`). In Letzteren tauchen wiederum andere Orte auf. Die Beispiel-DTD enthält also eine Rekursion. Rekursionen müssen aufgelöst werden, indem separate Relationen eingeführt werden. Im Beispiel wird `ort` nicht als Datenbankattribut von `ausflugsziel` definiert, sondern als Verweis auf eine eigene Relation `ort` mit eindeutiger ID. Hier ist das schon deshalb der Fall, weil `ort` das Wurzelelement des XML-Dokumentes ist. Wenn `ort` keine eigene Relation darstellen würde, müsste diese eingeführt werden.

Die Übersetzung von *DTDs in Datenbankentwürfe für objektrelationale DBS* und die Optimierung des resultierenden DB-Schemas wird in [KlMe00] beschrieben. Weiterhin wird in diesem Artikel betrachtet, welche zusätzlichen Übersetzungsschritte erforderlich sind, wenn relationale DBS als Basis für die Speicherung verwendet werden sollen. Es sei auch bemerkt, dass eine Abbildung auf objektrelationale Systeme von zunehmender Bedeutung ist, da der SQL:1999-Standard entsprechende Systemeigenschaften vorsieht. SQL:1999 ist im entsprechenden Standardisierungsdokument dokumentiert (vgl. [[ISO299]]) oder beispielsweise in [PeUn01]. Um die Abbildung von SQL:1999-Datentypen auf XML-Schema-Datentypen kümmert sich SQL/XML und die damit verbundenen Standardisierungsbemühungen (vgl. z.B. [SQLXML01] und [ZeMM01], [EiMe01]).

5.6.2 Abbildung auf relationale Datenbanksysteme

Wird anstatt eines objektrelationalen oder objektorientierten DBMS ein relationales verwendet, so gestaltet sich die Abbildung aufgrund der verstärkten Datenmodellheterogenität umständlicher. Eine Methode zur Abbildung der XML-Dokumentstrukturen auf das Schema eines relationalen DBS wurde in [STHZ99] vorgestellt. Das Vorgehen ist ähnlich dem im vorherigen Abschnitt vorgestellten. Zusätzlich muss hier die Übersetzung der Hierarchien in flache Relationen erfolgen. Dazu sind die gleichen Normalisierungsschritte notwendig, die bei der Über-

setzung von objektorientierten oder objektrelationalen Konzepten in relationale erforderlich sind.

Relationales DB-Schema für Hotelbeispiel. Die im vorherigen Abschnitt dargestellte DB ist nicht in erster Normalform, da die DTD *verschachtelte Elemente* enthält und diese als nicht atomare Attribute abgebildet wurden (das Attribut `adresse` mit `plz`, `ort` etc.). Nichtatomare Attribute müssen daher in eigene Relationen ausgegliedert und mit Schlüssel versehen werden. Die Bildung eigener Relationen ist bei relationalen DBS auch notwendig, wenn Elemente mit den Quantifizierern * oder + definiert werden (siehe Element `telefon`), da eine Abspeicherung als *Listen* wegen der dann gegebenen Verletzung der ersten Normalform nicht möglich ist. Für die Hotel-DTD entstehen dabei die in Abb. 5-11 dargestellten Relationen. Verbundattribute dienen dazu, die Zusammenhänge zwischen den Elementen auszudrücken.

Hotel

id	url	autor	hotelname	kategorie	adresse	telefon	fax	anreisebeschreibung
0001	www.hotel-huebner.de	Max	Hotel Hübner	4	a0001	t0001	0381/ 543444	Aus Richtung Rostock ...

Adresse

adresseID	plz	ort	strasse	nummer
a0001	18119	Warnemünde	Seestrasse	12

Telefon

TelefonID	telefon
t0001	0381/54340
t0001	0381/540

Abb. 5-11: *Relationales DB-Schema für das Hotelbeispiel*

In [DeFS99] wird ein Algorithmus vorgestellt, der auf Basis bestimmter, frei wählbarer Parameter ein relationales *DB-Schema automatisch generiert*. Parameter können dabei die maximale Anzahl abgeleiteter Relationen, der maximale Speicherplatzbedarf oder die maximale Anzahl von Attributen sein. In [KKRR00] werden eine Reihe von *Abbildungsmustern* für relationale DB vorgestellt, die aufgrund der Charakteristika von Elementen bzw. Attributen festlegen, ob eine Speicherung als Attribute in der gleichen Relation möglich ist oder ob andere Relationen verwendet werden müssen. Wenngleich der Hauptfokus auf einer benutzerdefinierten Abbildung liegt, wäre auf Basis der vorgestellten Muster auch eine Generierung des entsprechenden DB-Schemas möglich. Im Besonderen werden bei Elementen die Existenz von Attributen, Schachtelungsstrukturen, Kardi-

nalitäten und des #PCDATA-Datentyps betrachtet. Bei Attributen sind insbesondere Null- und Defaultwerte sowie Datentypen für die Abbildung bestimmend.

5.6.3 Einsatz benutzerdefinierter Abbildungsvorschriften

Neben den in den vorherigen Abschnitten vorgestellten Methoden, die eine automatische Abbildung realisieren, besteht auch die Möglichkeit, dass der Benutzer die Art der Abbildung individuell festlegt. Dabei gibt es unterschiedliche Möglichkeiten, die Abbildungsvorschriften zu repräsentieren. Sie können entweder innerhalb der Anwendung bzw. innerhalb der Anfragen, die den Zugriff durchführen, »*fix-verdrahtet*« oder davon *getrennt repräsentiert* werden.

Fix-verdrahtete Repräsentation. Ein Beispiel für fix-verdrahtete Repräsentation wird im Microsoft SQL-Server eingesetzt. Dort besteht die Möglichkeit, XML-Dokumente in ein DBS einzufügen. Dazu wird eine benutzerdefinierte Abbildung verwendet, die syntaktisch ein annotiertes XDR-Schema darstellt. XDR steht für *XML Data, subset reduced* und bezeichnet die von Microsoft entwickelte Schemasprache. Durch die speziellen Annotationen wird die Zuordnung zwischen XML-Dokument und relationalem DBS definiert. Die Zuordnungen lauten zum Beispiel `sql:relation` und `sql:field` und bestimmen die Zuordnung von Elementen und Attributen zu Datenbankrelationen und -attributen. Damit ist eine flexible Abbildung von XML-Dokumenten auf DBS möglich.

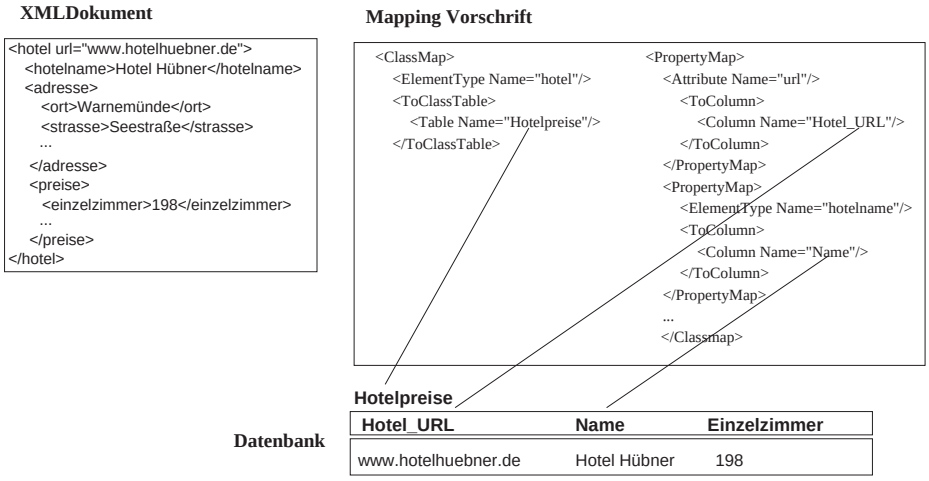


Abb. 5-12: *Repräsentation der Abbildungsvorschriften in einem XML-Dokument*

Separate Repräsentation. Werden Abbildungsvorschriften getrennt repräsentiert, können sie beispielsweise als XSLT-Regeln vorliegen oder als eigene XML-Dokumente. Abb. 5-12 zeigt eine Möglichkeit der Verwendung einer benutzerdefinierten Abbildungsvorschrift [BoBB00].

Das Element `hotel` wird auf eine Relation `Hotelpreise` abgebildet. Die Abbildung des Attributes `url` und des Blattelements `hotelname` erfolgt auf die Datenbankattribute `Hotel_URL` und `Name`.

Anfragen. Bei allen Verfahren, die eine Speicherung der XML-Dokumente in DBS vornehmen, ist es möglich, die gespeicherten XML-Dokumente durch Verwendung einer XML-Anfragesprache XPath oder XQuery zu erfragen. Voraussetzung dafür ist, dass die Abbildung von XML-Dokumenten auf die Datenbankstrukturen bekannt und nachvollziehbar ist. Die XML-Anfragen müssen bei dieser Speicherungsmethode von den jeweiligen eingesetzten Systemen auf Datenbankanfragen abgebildet werden. Aus den Ergebnissen der Datenbankanfragen müssen XML-Dokumente zusammengesetzt werden. Bei der Verwendung einer benutzerdefinierten Abbildung kann es möglich sein, dass die Anfragen nicht aus den gespeicherten Informationen beantwortet werden können, da die Abbildung nicht vollständig erfolgen muss.

Es ist darüber hinaus auch möglich, Anfragen direkt an das DBS zu stellen. In diesem Fall können SQL- oder OQL-Anfragen verwendet werden, um die Informationen zu erfragen und auszuwerten.

5.6.4 Bewertung der vorgestellten Methoden

Alle in diesem Kapitel vorgestellten Methoden weisen ähnliche Eigenschaften auf. Deshalb werden die Eigenschaften aller Methoden jetzt im Zusammenhang vorgestellt. Grundsätzlich ist eine Speicherung auf Schemaebene insbesondere für *datenorientierte XML-Dokumente* geeignet, die ein explizit deklariertes Schema aufweisen, auf dessen Basis der Datenbankentwurf bzw. die Abbildung zwischen den Schemata vorgenommen wird.

Nullwerte. Unregelmäßige XML-Dokumentstrukturen würden zu einer *hohen Anzahl an Nullwerten führen*. Inkonsistenzen und Performanzeinbußen wären die Folge. Dem könnte man zum Beispiel begegnen, indem alternative Elemente nicht auf eine einzelne Relation abgebildet, sondern als eigene Relationen ausgegliedert werden.

Elementordnung. Die *Elementordnung* aus den XML-Dokumenten geht bei dieser Speicherungsmethode verloren. Ein Ausweg ist die Einführung eines zusätzlichen Attributes, in dem die Ordnung gespeichert wird.

Anfragen. Bei diesen Speicherungsvarianten kann man die *SQL-Anfragefunktionalität* des DBS einsetzen. Es lassen sich z.B. Joins, Aggregatfunktionen usw. ausführen. Auch die Anfrageoptimierung übernimmt das DBMS. Prinzipiell ist es auch möglich, *XML-Anfragesprachen* zu verwenden und eine Abbildung auf SQL zu realisieren. Aus den Ergebnissen der Datenbankanfragen müssen dann die Ergebnisse für die XML-Anfrage gebildet werden.

Relational vs. objektrelational. Bei der Verwendung von relationalen DBS besteht insbesondere bei tiefen Schachtelungsstrukturen die Gefahr einer *exzessiven*

Fragmentierung des DB-Schemas, falls für ein Element, unabhängig davon ob es Subelemente, Attribute oder nur ein `#PCDATA` aufweist, eine entsprechende Relation erzeugt wird. Im Gegensatz dazu erlauben objektorientierte oder objektrelationale DBMS eine *natürlichere Abbildung der Elementhierarchien* und damit auch eine *einfachere Wiederherstellung*, wodurch sie sich für diese Speicherungsmethode besser eignen als relationale DBMS.

Abbildungsvorschriften. Eine separate Repräsentation der Abbildungsvorschriften zeichnet sich insbesondere durch Abbildungstransparenz beim Zugriff auf die gespeicherten XML-Dokumente und einfache Wartbarkeit der Abbildungsvorschriften aus, was besonders für benutzerdefinierte Abbildungen von Vorteil ist. Demgegenüber stehen mögliche Performanzvorteile bei einer fixen Verdrahtung.

Vorteile einer benutzerdefinierten Abbildung. Wird eine benutzerdefinierte Abbildung unterstützt, impliziert dies ein hohes Maß an Flexibilität. Einerseits kann der Benutzer die Abbildung *selektiv* vornehmen, indem die Bereiche der XML-Dokumentstruktur festgelegt werden, die abgebildet werden sollen. Andererseits kann der Benutzer die *Abbildungsvorschrift selbst festlegen*, um eine adäquate Form der Speicherung zu gewährleisten.

Darüber hinaus können die zu integrierenden Schemata (sowohl die XML-Dokumentstruktur als auch das DB-Schema) relativ *autonom entworfen* werden und damit an spezifische Anforderungen des Anwendungsbereichs angepasst werden. Erst die Abbildungsvorschrift ist für eine entsprechende Mediation zwischen den heterogenen Schemata zuständig. Das bedeutet zum Beispiel auch, dass eine Speicherung von XML-Dokumenten in bereits bestehende DBS erfolgen kann und so die Informationen aus den Dokumenten zusammen mit anderen Datenbankinformationen integriert oder föderiert werden können.

Nachteile einer benutzerdefinierten Abbildung. Ein gravierender Nachteil einer benutzerdefinierten Abbildung liegt darin, dass in hohem Ausmaß Schemaheterogenitäten auftreten können, was wiederum in komplexen Abbildungsvorschriften resultiert. Darüber hinaus müssen bei Änderungen der XML-Dokumentstruktur im Unterschied zur Speicherung auf Instanzebene entweder das DB-Schema oder aber die Abbildungsvorschriften entsprechend angepasst werden.

Das Erstellen von Abbildungsvorschriften erfordert einen großen *manuellen Aufwand*. Die Verantwortung für die »korrekte« Abbildung der Informationen liegt beim Entwerfer der Abbildungsvorschrift. Durch eine fehlerhafte Abbildung kann ein Informationsverlust verursacht werden, das heißt, dass Informationen, die im Originaldokument verfügbar waren, in dem DBS weder angefragt noch wiederhergestellt werden können. Die Informationen eines XML-Dokumentes sind daher auch nicht notwendigerweise aus der DB gemeinsam erfragbar, obwohl das bei einer direkten Anfrage an das XML-Dokument möglich wäre. Dies ist z.B. dann der Fall, wenn Elemente, die in einer Elementhierarchie vorkommen, in verschiedenen Relationen gespeichert werden, die nicht durch entsprechende Verbundattribute miteinander in Beziehung stehen.

5.7 Hybride Speicherung

Die in den vorangegangenen Abschnitten vorgestellten Speicherungsmethoden eignen sich jeweils für unterschiedliche Arten von Dokumenten. Einige lassen sich besser für datenorientierte, andere besser für dokumentorientierte XML-Dokumente einsetzen. Wie jedoch bereits in Abschnitt 5.2.1 ausgeführt, sind die Dokumenteigenschaften in vielen Fällen nicht so eindeutig feststellbar. Dokumente enthalten oft sowohl daten- als auch dokumentorientierte Anteile. Verwendet man zur Speicherung solcher gemischt strukturierten Dokumente eine der vorgestellten Methoden, wird man den einzelnen Dokumentanteilen nur zum Teil gerecht. Daher wurden *hybride Speicherungsverfahren* entwickelt, deren Grundidee darin besteht, gemischt strukturierte XML-Dokumente in daten- und dokumentorientierte Teile aufzusplitten und eine separate Speicherung mit der jeweils am besten geeigneten Speicherungsmethode vorzunehmen.

So kann es etwa bei der Speicherung von Hotels sinnvoll sein, die strukturierten Daten in bereits bestehende DBS zu integrieren, textuelle Beschreibungen dagegen als CLOBs zu speichern, über denen Indexe angelegt werden. Einerseits besteht damit die Möglichkeit, neben XML-Anfragen auch Anfragen zu realisieren, die aus dem Information Retrieval bekannt sind. Andererseits können DB-Anfragen zur Recherche in den strukturierten Daten herangezogen werden.

Abb. 5-13 veranschaulicht für das Hotelbeispiel den Einsatz von zwei Speicherungsmethoden. Dabei werden Hotelnamen und Adressen strukturiert in einem relationalen DBS gespeichert, während die Anreisebeschreibung als Volltext verwaltet wird.

hotelname	adresse				anreisebeschreibung
	ort	plz	strasse	nummer	
Hotel Hübner	Warne-münde	18119	Seestrasse	12	<auto>Aus Richung Rostock kommend ..</auto> <bahn>vom Bahnhof Warnemünde ... </bahn>

Abb. 5-13: *Hybride Speicherung von XML-Dokumenten*

Entscheidung über die Speichermethode. Es stellt sich nun die Frage, wie für solche Dokumente die Entscheidung getroffen wird, für welche Anteile welche Speicherungsmethode zu wählen ist. Eine Variante ist, dass ein Datenbankentwerfer oder XML-Anwender die einzelnen Anteile bestimmt. Durch das Wissen über die Semantik der einzelnen Elemente und Attribute kann er gut einschätzen, welche Speicherungsmethode sich besonders eignet. Eine andere Möglichkeit ist, diese Entwurfsentscheidung vollständig oder teilweise zu automatisieren. Insbesondere *Strukturinformationen* (aus DTDs oder XML Schema) sowie *Statistiken über Dokumente und Anfragen* liefern Hinweise auf die Wahl der Speicherungsmethode.

Ein Algorithmus, der aus strukturellen Merkmalen einer DTD sowie Statistiken über Dokumentkollektionen und Anfragen die daten- und dokumentorientierten Anteile eines XML-Dokumentes ermittelt, wurde in [KlMe00] vorgestellt. Dieser Algorithmus ist Voraussetzung zur Bestimmung einer hybriden Speicherung von XML-Dokumenten.

Ein ähnliches Herangehen wird in dem Artikel von Deutsch et al. [DeFS99] beschrieben. Im Mittelpunkt des dort vorgestellten Systems STORED steht eine Abbildungsvorschrift, die beschreibt, wie XML-Dokumente auf relationale DBS abgebildet werden. Die Abbildung kann sowohl vom Benutzer angegeben werden als auch durch einen eigens dafür entwickelten Algorithmus mit Hilfe von Text-Mining-Methoden ermittelt werden. Bei den Text-Mining-Methoden wird aus einer Beispieldokumentkollektion ermittelt, welche Elemente und Attribute in engem Zusammenhang stehen. Die Abbildungsvorschrift wird dann so generiert, dass diese gemeinsam auf Relationen der DB abgebildet werden. Für sehr selten auftretende Elemente wird anstelle der strukturierten Speicherung in dem DBS eine Speicherung als Graph eingesetzt.

5.8 Zusammenfassung und Ausblick

In diesem Kapitel wurden verschiedene Methoden zur Speicherung von XML-Dokumenten vorgestellt. Abb. 5-14 stellt die vorgestellten Verfahren (ohne hybride Speicherung) noch einmal zusammen und gibt an, für welche Arten von XML-Dokumenten die einzelnen Methoden besonders geeignet sind.

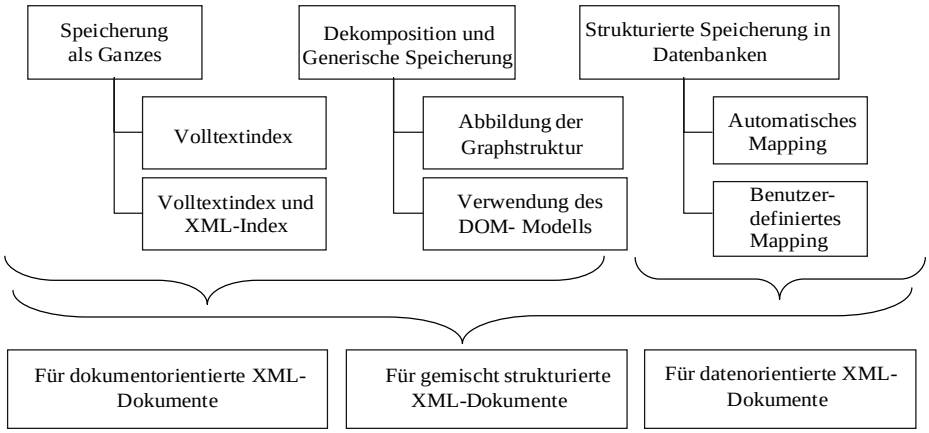


Abb. 5-14: *Verwendung der Speicherungsverfahren*

Speicherung als Ganzes. Methoden zur Speicherung von XML-Dokumenten in Dateien verändern die Dokumente selbst nicht. Die Dokumente werden indexiert, im Index werden Informationen über Inhalte und Strukturen der XML-Dokumente dargestellt. Diese Methode eignet sich besonders für *dokumentorientierte* XML-Dokumente.

Dekomposition und generische Speicherung auf Instanzebene. Diese Methode der Speicherung (Abschnitt 5.5) bietet eine einfache Möglichkeit, die Dokumentstruktur generisch zu speichern. Vorteile sind hier die Unabhängigkeit von der Existenz einer expliziten Beschreibung der XML-Dokumentstruktur sowie das feststehende Datenbankschema für alle Dokumentklassen und die daraus resultierende Einfachheit der Abbildungsvorschriften. Nachteilig ist die umständliche Realisierung von komplexeren Anfragen.

Strukturierte Speicherung in Datenbanken auf Schemaebene. Diese Methode der Speicherung (Abschnitt 5.6) ermöglicht eine adäquate Darstellung von XML-Dokumenten in objektorientierten, objektrelationalen oder relationalen DBS. Hierbei ist das Datenbankschema von der DTD (oder einer anderen explizit deklarierten Struktur wie XML Schema) durch Definition entsprechender Abbildungsvorschriften abhängig. Die Methode ist insofern aufwändiger, da für jede neue Dokumentkollektion ein neues Datenbankschema gebildet werden muss. Eine Änderung des Schemas der XML-Dokumente bewirkt eine Änderung des Datenbankschemas.

Dieser Prozess der Generierung eines DB-Schemas ist automatisierbar. Durch benutzerdefinierbare Abbildungsvorschriften wird ein hohes Maß an Flexibilität erreicht. Es besteht jedoch bei einer inkorrekten Abbildung die Gefahr des Informationsverlustes. Diese Methode eignet sich nur für XML-Dokumente mit explizit vorhandenem und relativ unveränderlichem Schema, also nur für *datenorientierte* XML-Dokumente.

Hybride Speicherung. Liegen Dokumente vor, in denen sowohl daten- als auch dokumentorientierte Anteile enthalten sind, so kann eine Aufsplittung der Dokumente und eine Speicherung der einzelnen Anteile mit verschiedenen Methoden sinnvoll sein.

Parallele Verwendung mehrerer Speicherungsmethoden. Weiterhin können auch mehrere Methoden gleichzeitig für die Speicherung von XML-Dokumenten verwendet werden. Diese Redundanz ist sinnvoll, wenn die Informationen aus den Dokumenten auf sehr verschiedene Weise genutzt werden sollen. Wenn zum Beispiel XML-Dokumente einerseits für Suchoperationen bereitgestellt werden sollen, bei denen die vollständigen Dokumente als Ergebnis zurückgeliefert werden sollen und andererseits komplexe Auswertungen zum Beispiel durch Verwendung von Aggregatfunktionen durchgeführt werden sollen, so sind hier verschiedene Methoden geeignet. Es ist in solchen Fällen durchaus sinnvoll, für die XML-Dokumente redundant mehrere Speicherungsmethoden zu verwenden. Beispielsweise kann es sinnvoll sein, einerseits eine vollständige Speicherung und Indexierung der XML-Dokumente vorzunehmen, wodurch eine einfache Wiederherstellbarkeit geboten ist, und parallel dazu ein Verfahren einzusetzen, das die Inhalte der XML-Dokumente in DBS speichert.

Anfragen an gespeicherte XML-Dokumente sind bei allen vorgestellten Verfahren mit XPath möglich. Prinzipiell können auch XQuery-Anfragen für alle Vari-

anten eingesetzt werden. XQuery zeichnet sich zum Zeitpunkt des Erscheinens dieses Buches erst als erste Empfehlung des W3C für eine Anfragesprache ab. Aufgrund dessen basieren alle Ansätze aus Forschung und Industrie bisher auf vorläufigen Versionen von XQuery. Hier wird es also in Zukunft noch Änderungen geben. Damit verändert sich auch die Anfragerealisierung auf den entsprechenden Speicherungsstrukturen.

Da es bislang keine als Empfehlung verabschiedete Anfragesprache für XML-Dokumente gibt, gibt es auch noch keine *Update-Sprache*. In XQuery 1.0 wird es keine Änderungsoperationen geben. Es ist jedoch nicht auszuschließen, dass Updates in zukünftigen Versionen ergänzt werden. Erste Vorschläge zur Erweiterung können in [TIHW01] und [RoCF01] nachgelesen werden. Änderungsoperationen an XML-Dokumenten werden bislang über die DOM-API realisiert oder durch Transformationen mittels XSLT umgesetzt. Das System eXcelon bietet mit XUL (XML Update Language) eine eigene Änderungssprache an. XUL stellt eine Erweiterung von XSLT dar. So genannte »Updategrams« sind XML-Instanzen, die Folgen von Navigationen auf der Baumstruktur und Änderungsoperationen darstellen. Die Notwendigkeit von Updates besteht auch für XML-Dokumente. Bisher ist es jedoch nur möglich, Methoden, die im Document Object Model definiert wurden, zum Update zu verwenden. Wenn zukünftig Sprachen zur Darstellung von Updates existieren, so muss für jede Speicherungsvariante die Möglichkeit geschaffen werden, Updates entsprechend durchzuführen.

In diesem Kapitel haben wir dargestellt, dass verschiedene Speicherungsmethoden sich für unterschiedliche Arten von XML-Dokumenten eignen. Bisher gibt es noch keinen Vergleich der Effizienz dieser Methoden. Ein *Benchmark*, der zum Vergleich verwendet werden kann, wurde von Böhme/Rahm [BöRa01] entwickelt. Er wird zusammen mit dem Stand der Arbeiten auf dem Gebiet in Kapitel 14 vorgestellt.

Literatur

[BoBB00] Bourret, R., Bornhövd, C., Buchmann, A.P.: *A Generic Load/Extract Utility for Data Transfer Between XML Documents and Relational Databases*. 2nd Int. Workshop on Advanced Issues of EC and Web-based Information Systems (WECWIS), San Jose, California, June, 2000.

[BöRa01] Böhme, T., Rahm, E.: *XMach-1: A Benchmark for XML Data Management*. BTW 2001, S. 264-273.

[BöTü02] Böttcher, S., Türling, A.: *Umkehrbares Mapping von XML-Updates auf relationale Datenbankinhalte*, Datenbank-Spektrum, dpunkt.verlag, 2/2002.

[Bour02] Bourret, R.: *XML Database Products – Native XML Databases*. http://www.rpbourret.com/xml/ProdsNative.htm, 2002

[Brad98] Bradley, N.: *The XML companion*, Addison Wesley, 1998.

[Carl01] Carlson, D.: *Modeling XML Applications with UML*, Addison-Wesley, 2001.

[DeFS99] Deutsch, A., Fernandez, M., Suciu, D.: *Storing Semistructured Data with STORED*. ACM SIGMOD Record, Vol. 28, No. 2, June 1999.

[EdHo00] Edwards, R., Hope, S.: *Persistent DOM: An Architecture for XML Repositories in Relational Databases*, Intelligent Data Engineering and Automated Learning – IDEAL, 2000.

[EiMe01] Eisenberg, Andrew; Melton, Jim: *SQL/XML and the SQLX Informal Group of Companies*; Whitemarsh Information Systems Cooperation: www.wiscorp.com

[FiKM01] Fiebig, T., Kanne, C.C., Moerkotte, G.: *Natix – ein natives XML-DBMS*, Datenbank-Spektrum, Volume 1, Heft 1, S.5-13, 2001.

[Fink02] Finkelstein, L., et al.: *Placing Search in Context: The Concept Revisited*, ACM Transactions on Information Systems, Vol. 20, No. 1, Jan. 2002.

[FlKo99] Florescu, D., Kossmann, D.: *Storing and Querying XML Data Using an RDBMS*. IEEE Data Engineering Bulletin, Special Issue on XML, Vol. 22, No. 3, September, 1999.

[Gold90] Goldfarb, C. Rubinsky, Y. (Contributor): *The SGML Handbook*. Oxford University Press, Oxford, UK, 1990.

[Haro01] Harold, E.R., Means, W.S.: *XML in a Nutshell*, O'Reilly, 2001.

[HiKa99] Hitz, M., Kappel, G.: *UML @ Work*, dpunkt.verlag, 1999.

[ISO299] ISO-ANSI: *Information Technology – Database Languages – SQL – Part 2: Foundation (SQL/Foundation)*; ISO/IEC 9075-2:1999.

[KKRR00] Kappel, G., Kapsammer, E., Rausch-Schott, S., Retschitzegger, W.: *X-Ray – Towards Integrating XML and Relational Database Systems*. Proc. of the 19th Int. Conf. on Conceptual Modeling (ER), LNCS 1920, Springer, Salt Lake City, USA, Oct., 2000.

[KlLi02] Kleiner, C., Lipeck, U.W.: *Automatische Erzeugung von XML-DTDs aus konzeptionellen Datenbank-Schemata*, Datenbank-Spektrum, dpunkt.verlag, 2/2002.

[KlMe00] Klettke, M., Meyer, H.: *XML and Object-Relational Database Systems – Enhancing Structural Mappings Based on Statistics*. Int. Workshop on the Web and Databases (WebDB), Dallas, May, 2000.

[KlMe02] Klettke, M., Meyer, H.: *XML und Datenbanken*. dpunkt.verlag, 2003.

[LaMa01] Lausen, G., Marron, P.J., *On Processing XML in LDAP*, Proc. of the 27th Int. Conf. On Very Large Databases (VLDB), 2001.

[PeUn01] Pernul, Günther; Unland, Rainer: *Datenbanksysteme in Unternehmen: Analyse, Modellbildung und Einsatz*; Oldenbourg Verlag; 2001.

[PrRW01] Pröll, B., Retschitzegger, W., Wagner, R.: *TIScover – Eine generische Plattform für webbasierte Tourismusinformationssysteme*, Informatik Forschung und Entwicklung, Themenheft Electronic Commerce, Springer Verlag, Vol. 16, Issue 1, 2001.

[RoCF01] Robie, J., Chamberlin, D., Florescu, D.: *XQuery – The W3C XML Query Language*, Software AG Experience 2001.

[SaGi83] Salton, G., McGill, M.: *Introduction to Modern Information Retrieval*, McGraw-Hill, New York, NY, 1983.

[ScCo01] Scheffner, D., Conrad, R.: *Access Support Tree & TextArray: A Model for Physical Storage of XML Documents*, GI-Workshop Web-Datenbanken, 2001.

[ShYU99] Shimura, T., Yoshikawa, M., Uemura, S.: *Storage and Retrieval of XML Documents Using Object-Relational Databases*, Proc. of the Int. Conf. On Database and Expert System Applications (DEXA), 1999.

[SQLXML01] *XML-Related Specifications (SQL/XML)* – Working Draft SQL:200n Part 14, H2-2001-149, WG3:YYJ-012, Jim Melton (Editor), June 18, 2001, available at http://www.sqlx.org.

[STHZ99] Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., DeWitt, D., Naughton, J.: *Relational Databases for Querying XML Documents – Limitations and Opportunities*. Proc. of the 25th Int. Conf. On Very Large Data Bases (VLDB), Edinburgh, 1999

[TIHW01] Tatarinow, I., Ives, Z. G., Halevy, A. Y., Weld, D. S.: *Updating XML*. In SIGMOD Conference 2001.

[Trup00] Trupin, J.: *SQL Server 2000: New XML Features Streamline Web-centric Application Development*, Technical Report, Microsoft Corp., 2000.

[UML01] *Unified Modeling Language (UML), version 1.4*, OMG, 2001, http://www. omg.org/technology/documents/formal/uml.htm.

[W3C02] *Document Object Model*, World Wide Web Consortium (W3C), 2002, http://www.w3c.org/DOM.

[YASU01] Yoshikawa, M., Amagasa, T., Shimura, T., Uemura, S., *XRel: A Path-Based Approach for Storage and Retrieval of XML Documents Using Relational Databases*: ACM Transactions on Internet Technology, Volume 1, Nummer 1, S. 110-141, 2001.

[ZeMM01] Zemke, Fred; Malhotra, Ashok; Melton, Jim: *Mapping SQL types to XML types – an overview*; Whitemarsh Information Systems Cooperation: http://www.sqlx.org.

6 Datenintegration und Mediatoren

Kai-Uwe Sattler, Stefan Conrad, Gunter Saake

Kurzfassung

Die Integration verteilter, heterogener Datenbestände stellt gerade im World Wide Web mit der Vielzahl verfügbarer Quellen eine besondere Herausforderung dar. Gegenstand dieses Kapitels sind daher Techniken der virtuellen Integration unter Verwendung so genannter Mediatoren, die einen effizienten und aktuellen Zugriff auf weltweit verteilte Daten sowie deren Kombination ermöglichen. Im Mittelpunkt stehen dabei Aspekte der Architektur solcher Systeme sowie die Überwindung der Heterogenität auf Schema- und Datenebene. Weiterhin werden Fragen der effizienten Verarbeitung von Anfragen behandelt, die spezielle Optimierungstechniken ebenso einschließen wie Techniken zur Anfrageausführung unter den besonderen Bedingungen des Web.

6.1 Einleitung: Datenintegration im Web

Die Entwicklung des World Wide Web hat in den letzten Jahren den Zugriff auf weltweit verteilte Informationsquellen erheblich vereinfacht. Web-basierte Benutzerschnittstellen verbergen komplexe Anfragesprachen und -protokolle, während Suchmaschinen scheinbar jede gewünschte Information erschließen können. Dennoch ist es oftmals schwierig, aufgrund der riesigen Menge an verfügbaren Daten relevante Inhalte zu identifizieren – Suchmaschinen liefern auf eine Anfrage meist Tausende von Dokumenten, die den Suchbegriff an irgendeiner Stelle enthalten. Darüber hinaus erschwert die Verwendung von HTML als »Datenaustauschformat« die Kombination von Informationen aus verschiedenen Quellen, beispielsweise um den günstigsten Preis eines Produktes aus mehreren Online-Shops zu ermitteln, die relevante Literatur zu einem bestimmten Gebiet abzufragen oder im aktuellen Fernsehprogramm weitere Informationen zu einem Film zu suchen.

Zur Beantwortung solcher Anfragen müssen zwei wesentliche Voraussetzungen erfüllt sein:

- Es sind *strukturierte Anfragen* zu unterstützen, d.h. Anfragen über konkrete, identifizierbare Eigenschaften von Objekten wie etwa Preis, Bezeichnung o.ä., im Gegensatz zur Stichwortsuche typischer Volltextsuchmaschinen.

- Verschiedene, für den gegebenen Anwendungsbereich relevante Quellen müssen *integriert* werden. Dies bedeutet, einerseits eine homogene Sicht bezüglich Schnittstellen und Struktur auf alle diese Quellen bereitzustellen und andererseits für Transparenz hinsichtlich der Herkunft der Daten zu sorgen.

Die Aufgabe besteht demnach darin, Anfragen zu beantworten, indem Daten aus externen, heterogenen Quellen extrahiert und kombiniert werden. So könnte man z.B. das aktuelle Fernsehprogramm über die Website des Senders ermitteln, die Filme im Programm in der Internet Movie Database www.imdb.org zusammen mit einigen Reviews suchen, aus einer Online-Enzyklopädie weitere Informationen zu den Schauspielern beziehen und schließlich die Trailer von der Website zum Film laden.

Ein wesentliches Problem in diesem Kontext stellt die Heterogenität der Quellen dar, die sich auf verschiedenen Ebenen äußert:

- auf Systemebene in Form unterschiedlicher Anfrageschnittstellen,
- auf Datenmodellebene durch die Verwendung von HTML, XML, relationaler oder objektorientierter Schemata,
- auf Schemaebene durch unterschiedliche Modellierung des Weltausschnittes und schließlich
- auf Datenebene durch verschiedene Repräsentationen ein und desselben Sachverhaltes.

Somit bestehen für derartige Web-Integrationssysteme große Ähnlichkeiten zu heterogenen Datenbanksystemen wie Multidatenbanken oder föderierten Datenbanken. Datenintegration im Web ist aber darüber hinaus mit folgenden Besonderheiten verbunden:

- eine meist große Anzahl von Quellen, die auch teilweise noch häufig wechseln,
- eine hohe Autonomie der Quellen hinsichtlich der Verfügbarkeit oder der Änderung der Schemata,
- bei den Quellen handelt es sich oft um keine voll-funktionalen Datenbanksysteme oder die Datenbankfunktionalität ist bewusst hinter einer einschränkenden Web-Schnittstelle verborgen, die nur bestimmte Anfragen zulässt,
- es liegen nur wenige Informationen über die Eigenschaften der Quellen vor, z.B. zum Inhalt, zu den operationalen Fähigkeiten oder zu den Kosten von Operationen.

Zur Realisierung von integrierten Systemen für Web-Daten lassen sich zwei grundlegende Varianten unterscheiden:

- Beim Ansatz der *Materialisierung* werden die Daten periodisch aus den Quellen extrahiert und in einer zentralen Datenbank (ähnlich einem Data Warehouse) abgelegt. Dadurch werden Anfragen ausschließlich über den Daten der zentralen Datenbank beantwortet – zum Anfragezeitpunkt ist kein Zugriff auf die Quellsysteme mehr notwendig. Das Ergebnis ist eine hohe Performanz und die Möglichkeit, eine umfassende Datenbereinigung zur Beseiti-

gung von Inkonsistenzen und Fehlern in den Daten vornehmen zu können. Diese Vorteile werden jedoch mit Aktualitätsproblemen erkauft, d.h., es kann nicht gewährleistet werden, dass immer die aktuellen Informationen der Quelle als Antwort geliefert werden.

- Der *virtuelle* Ansatz basiert dagegen auf der Definition von Sichten über den Daten der Quellen. Demzufolge müssen Anfragen auf den integrierten Bestand in Anfragen an die Quellrelationen transformiert, an die Quellen gesendet und dort ausgeführt werden. Die Vorteile dieses Verfahrens sind die hohe Aktualität der Daten und das Wegfallen einer redundanten Datenhaltung. Demgegenüber stehen die Nachteile einer aufwändigen Anfrageausführung und die Abhängigkeit von der Verfügbarkeit der Quellen.

Beiden Ansätzen gemeinsam ist jedoch die Lösung von Problemen der Datenextraktion aus den Quellen sowie die Überwindung von Integrationskonflikten (siehe Abschnitt 6.3). Hinsichtlich der Anfragebearbeitung beruht der Ansatz der Materialisierung im Wesentlichen auf traditionellen Techniken zentraler Datenbanksysteme, während für virtuelle Ansätze spezielle Methoden benötigt werden (siehe Abschnitt 6.4).

Betrachtet man virtuelle Ansätze bezüglich der Art der zu integrierenden Daten und der unterstützten Operationen näher, so lässt sich eine Klassifikation wie in Abb. 6-1 angeben.

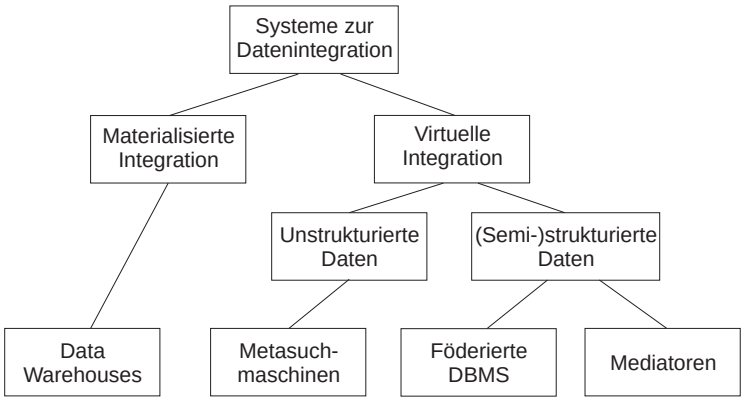


Abb. 6-1: *Klassifikation von Systemen zur Datenintegration*

Danach kann grob unterschieden werden zwischen Systemen, die eher unstrukturierte Anfragen (Volltextsuche) unterstützen, (z.B. Suchmaschinen und Metasuchmaschinen) sowie Systemen für strukturierte Anfragen. Zur letzteren Gruppe gehören neben föderierten Datenbanksystemen, die dem Benutzer eine weitgehend vollständige Datenbanksystemsicht inklusive Update-Operationen und Transaktionen anbieten, insbesondere auch Mediatoren, die nur Leseoperationen (Anfragen) unterstützen und meist auf einem semistrukturierten Datenmodell basieren.

Obwohl der Übergang zwischen beiden Ansätzen durchaus fließend ist, werden wir uns im Weiteren auf den Mediator-Ansatz konzentrieren, da dieser aufgrund der Eigenschaften die Anforderungen der Integration von Web-Daten besser erfüllt. Im folgenden Abschnitt 6.2 werden wir daher zunächst die Architektur von Mediatorsystemen näher betrachten. Die mit der Datenintegration in Mediatoren verbundenen Aufgaben und Probleme werden in Abschnitt 6.3 diskutiert. Gegenstand von Abschnitt 6.4 bilden Techniken der Anfragebearbeitung in Mediatoren, beginnend bei der Beschreibung von Quelleneigenschaften über Optimierungstechniken bis hin zur Ausführung der Anfragen von Anfragen. Schließlich werden in Abschnitt 6.5 die wesentlichen Erkenntnisse dieses Kapitels zusammengefasst.

6.2 Architektur von Mediatorsystemen

Nach der Vorstellung der prinzipiellen Varianten von Systemen zur Datenintegration in Abschnitt 6.1 wollen wir nun den Ansatz der virtuellen Integration genauer betrachten und dabei den Begriff des Mediators vertiefen. Die Idee von Mediatoren im Kontext der Datenintegration geht auf Wiederhold [Wied92] zurück und bezeichnet in einer sehr allgemeinen Form eine Softwarekomponente zur Vereinfachung, Reduzierung, Kombination und Erklärung von Daten. Ein Mediator bildet damit die zentrale Komponente in einer Architektur aus drei Ebenen (Abb. 6-2).

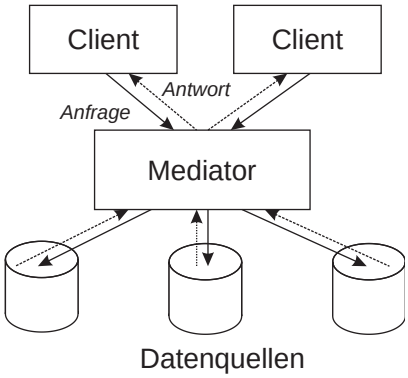


Abb. 6-2: *Mediator-Architektur nach Wiederhold*

Die unterste Ebene umfasst Datenquellen oder genauer Server, die Daten über Anfragen bereitstellen. Die mittlere Integrationsebene wird durch den Mediator gebildet, der ein globales Schema über alle Quellen anbietet, auf dessen Basis deklarative Anfragen formuliert werden können. Diese Anfragen werden durch die Clients der dritten Ebene in Form von Applikationen oder Nutzern gestellt.

Die Besonderheit von Mediatoren im Vergleich zu einem Data-Warehouse-Ansatz ist, dass in der mittleren Ebene keine Nutzdaten gespeichert sind. Dies hat

zur Folge, dass Anfragen an den Mediator durch Anfragen an die Quellsysteme beantwortet werden müssen. Die wesentlichen Aufgaben eines Mediators sind daher

- die Transformation von Anfragen an das globale Schema in Anfragen an Quellen sowie
- das Einsammeln und Verknüpfen der Ergebnisse.

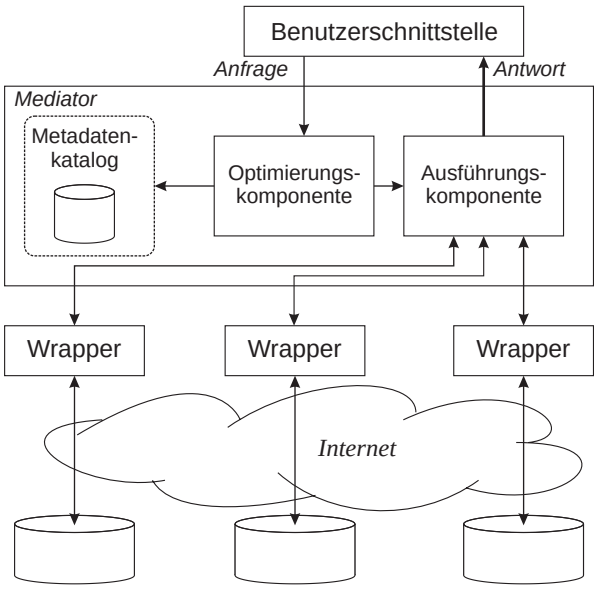


Abb. 6-3: *Komponenten eines Mediatorsystems*

Ein detaillierter Überblick zur typischen Architektur eines Mediatorsystems ist in Abb. 6-3 gegeben. Danach umfasst ein Mediator drei Hauptkomponenten [FlLM98]:

- Der *Metadatenkatalog* besteht aus dem globalen Schema (der »Weltsicht«), dessen Abbildung auf die lokalen Schemata der Quellen sowie einer Beschreibung der Quellen. Dem globalen Schema liegt ein geeignetes Datenmodell zugrunde – etwa (objekt-)relational oder semistrukturiert unter Verwendung von XML. Die Abbildungen zwischen den Schemata werden meist in Form von Sichtdefinitionen deklarativ beschrieben, wobei entweder das globale Schema als integrierte Sicht auf den lokalen Schemata der Quellen definiert wird (»Global-as-View«) oder die Inhalte der Quellen als Sicht (Ausschnitt) über den Gesamtdatenbestand spezifiziert werden (»Local-as-View«). Weiterhin ist zu jeder Quelle definiert, welche Daten (d.h. welcher Ausschnitt der globalen Sicht) enthalten sind. Teilweise werden auch noch die konkreten Anfragefähigkeiten der Quellen angegeben.

- Die *Planungs- oder Optimierungskomponente* ist für die Erstellung eines Plans – d.h. einer konkreten Ausführungsbeschreibung – zu einer gegebenen Anfrage verantwortlich. Hierzu wird die Anfrage zunächst in eine geeignete interne Repräsentation überführt. Anschließend erfolgt das Umschreiben oder Zerlegen der globalen Anfrage in durch die Quellsysteme ausführbaren Teilanfragen. Da es meist mehrere mögliche Pläne zu einer Anfrage gibt, müssen verschiedene Varianten untersucht und schließlich die effizienteste ausgewählt werden.
- Die *Ausführungskomponente* arbeitet den von der Optimierungskomponente erstellten Plan ab, indem die Teilanfragen an die jeweiligen Quellsysteme gesendet und die Ergebnisse eingesammelt werden. Diese Teilergebnisse werden entsprechend der Definition des globalen Schemas und der Anfrage kombiniert und dem Client zur Verfügung gestellt. Darüber hinaus werden auch noch Operationen (z.B. Selektionen) ausgeführt, die von einer Quelle aufgrund eingeschränkter Anfragefunktionalität nicht ausgeführt werden können.

Auf die mit der Optimierung und Ausführung von Anfragen in Mediatoren verbundenen Fragestellungen werden wir in Abschnitt 6.4 noch genauer eingehen.

Eine weitere wichtige Komponente in der Architektur bilden so genannte *Wrapper*, die eine Kopplung zwischen Mediator und Quellsystem vornehmen. Wrapper verbergen die Spezifika der Quellen bezüglich Datenmodell und Anfrageschnittstelle und bieten dem Mediator somit einen einheitlichen Zugriff. Zu den konkreten Aufgaben eines Wrappers gehören demnach:

- die Abbildung zwischen dem Datenmodell des Mediators und dem spezifischen Modell der Quelle,
- die Übersetzung der vom Mediator initiierten Anfrage in die Anfragesprache oder auch Funktionsaufrufe der Quellsysteme.

Während diese Aufgaben für relationale oder XML-basierte Datenbanksysteme durch die Verfügbarkeit von Standardschnittstellen einfach lösbar sind, stellen die typischen Web-Quellen mit Aufrufschnittstellen in Form von CGI-Programmen und Formularen sowie HTML als Ausgabeformat ein wesentlich größeres Problem dar. Ein Wrapper für eine solche Quelle muss zunächst die Anfrage vom Mediator in einen konkreten CGI-Aufruf umsetzen und anschließend das HTML-Dokument analysieren, um die eigentlichen Daten zu extrahieren. Der Aufwand zur Erstellung solcher Wrapper kann jedoch durch Toolkits oder Generatoren reduziert werden. Hierbei lassen sich zwei grundsätzliche Vorgehensweisen unterscheiden [FlLM98]:

- Bei den grammatikbasierten Ansätzen werden im Prinzip Parser zur Analyse der Ergebnisdokumente erstellt, die die relevanten Teile mit den eigentlichen Daten identifizieren und diese extrahieren.
- Bei der zweiten Klasse von Verfahren wird versucht, eine solche Grammatik durch Anwendung von Lernverfahren abzuleiten. Hierzu werden in repräsen-

tativen Beispielseiten die relevanten Dateneinträge markiert und dem Lernverfahren vorgelegt.

Dennoch kann die Entwicklung von Wrappern gerade für reine HTML-Quellen sehr aufwändig sein, insbesondere auch deshalb, weil Änderungen an den Quellen (hinsichtlich Layout der Ergebnisseiten, Gestaltung der Anfrageschnittstellen etc.) immer auch Anpassungen der Wrapper notwendig machen.

Ein weiterer wichtiger Aspekt der Architektur von Mediatorsystemen ist die Gestaltung der Schnittstelle zwischen Mediator und Wrapper. Dies betrifft nicht nur die Frage der Verwendung von Standardschnittstellen und -sprachen, sondern insbesondere auch die Fähigkeiten der Wrapper. So können leichtgewichtige (»thin«) Wrapper eingesetzt werden, die nur eine reine Transformation der Anfragen vornehmen. In diesem Fall muss auf Mediatorebene sichergestellt werden, dass die dem Wrapper übergebene Anfrage den Fähigkeiten der Quelle entspricht. Schwergewichtige oder »fat« Wrapper können dagegen auch Anfragen auswerten, die von der Quelle selbst nicht unterstützt werden. Hierbei sind die entsprechenden Operationen vom Wrapper zu implementieren. Obwohl die letztere Form von Wrappern die Erstellung von Mediatoren sowie die Anfragebearbeitung deutlich vereinfacht, ist jedoch ein deutlich höherer Entwicklungs- und Wartungsaufwand für die Wrapper erforderlich.

Die in Abb. 6-3 dargestellte Struktur entspricht einer typischen Architektur eines Mediatorsystems. Konkrete Systeme weisen im Detail jedoch Besonderheiten auf, deren mögliche Ausprägungen im Folgenden am Beispiel der Systeme TSIMMIS und KIND kurz aufgezeigt werden sollen.

Eines der ersten Mediatorsysteme war das an der Stanford University entwickelte TSIMMIS-System. TSIMMIS verwendet als internes Datenmodell das Object Exchange Model (OEM) – ein semistrukturiertes Datenmodell, das Objekte in einer selbstbeschreibenden Form repräsentiert, ohne dass dazu ein (externes) Schema notwendig ist. Ein OEM-Objekt ist dabei ein Tupel *(oid, label, type, value)*, bestehend aus einem Objektidentifikator, einen klassenähnlichen Bezeichner, einem Typ (atomar oder Menge) und einem entsprechenden Wert. Ein Archiv mit Kinofilmen, das jeweils Titel, Genre und Erscheinungsjahr umfasst, könnte demnach durch folgende Struktur repräsentiert werden:

```
<#obj1, archiv, set, {#obj2, … }>
  <#obj2, film, set, {#sub1, #sub2, #sub3}>
    <#sub1, titel, string, «Herr der Ringe«>
    <#sub2, genre, string, «Fantasy«>
    <#sub3,  jahr, int, 2001>
```

Mediator und Wrapper werden mit Hilfe von Regeln in MSL – einer logik-basierten Anfragesprache für OEM – spezifiziert und über Generatoren implementiert. Eine MSL-Regel besteht aus einem Regelkopf, der das Anfrageergebnis in Form von Mediatorobjekten beschreibt, und einem Regelkörper mit der Angabe der Bedingungen, die die Quellobjekte erfüllen müssen. Ein Beispiel für eine solche

Regel ist die folgende Anfrage an die Quelle source, die alle Titel von Fantasy-Filmen liefert:

```
<fantasy-filme T> :-      // Regelkopf
    <archiv { <film {<titel T>     // Regelkörper
                    <genre «Fantasy«>}> }> @source
```

In dieser Notation werden durch »<>« Objektlabel und zugehöriger Wert gruppiert, »{}« bezeichnet Mengen von Objekten und Großbuchstaben wie »T« stehen für Variablen, an die Werte gebunden werden. Die obige Anfrage ermittelt demnach ausgehend vom Archiv-Objekt alle Filme, die auf ein Genre-Objekt mit dem Wert »Fantasy« verweisen, und liefert die Titel-Werte zurück. Die Ähnlichkeit mit Pfadausdrücken in XML-Anfragen ist nicht zufällig: Tatsächlich haben die Arbeiten zu OEM die Entwicklung von XML-Anfragesprachen nachhaltig beeinflusst.

Eine globale Sicht des Mediators wird nun durch eine Menge von Regeln definiert, die beschreiben, wie die globalen (virtuellen) Objekte aus den Objekten der Quellen zusammengestellt werden.

Als Anfragesprache für Nutzer und Anwendungen bietet TSIMMIS darüber hinaus noch die OQL-ähnliche Sprache LOREL an. Weitere Details zu TSIMMIS sind u.a. in [PaGW95,VaPa97] zu finden.

Die mit TSIMMIS realisierte Form eines Mediators – die beinahe schon als »klassisch« bezeichnet werden kann – ist am besten für Anwendungen geeignet, wo die Quellen zu einer gemeinsamen, homogenen Domäne gehören und so die Extensionen bzw. Relationen durch relativ einfache Integrationsoperationen wie Verbund oder Vereinigung kombiniert werden können. In Integrationsszenarien mit eher disjunkten Domänen, etwa bei der Integration von Quellen mit Daten aus wissenschaftlichen Experimenten (z.B. in den Bio- und Neurowissenschaften), die verschiedene Aspekte repräsentieren (beispielsweise Zellstrukturen, Zellreaktionen etc.), sind dagegen oft eine Interpretation der Daten und die Verwendung von Hintergrundwissen notwendig, um die indirekten Beziehungen der Daten zu erkennen und auszunutzen.

Ein Beispiel eines Mediatorsystems, das eine solche »wissensbasierte« Integration realisiert, ist das KIND-System [LuGM01] für die Integration von Quellen aus dem Bereich der Neurowissenschaften. Der KIND-Mediator verwendet als internes Datenformat XML, verfügt aber darüber hinaus über so genannte *Domain Maps*, die semantische Netze aus Konzepten (Klassen), deren Beziehungen in Form von Spezialisierung/Generalisierung sowie Kompositionsbeziehungen und anwendungsspezifische Constraints darstellen. Die Konzepte bilden dabei die semantischen »Anker« für die zu integrierenden Daten. Eine Domain Map wird in Form von Fakten und logischen Regeln spezifiziert, wobei in der konkreten Implementierung F-Logic – eine objektorientierte Erweiterung von Datalog – als formale Basis verwendet wird.

Zur eigentlichen Datenintegration werden die Objekte der Quellen den Konzepten der Domain Map als Instanzen zugewiesen. Die Domain Map kann dabei

auch erweitert werden, indem neue Konzepte eingefügt und in Beziehung zu existierenden Konzepten gesetzt werden. Eine integrierte globale Sicht entspricht schließlich einer Klassendefinition. Diese kann nicht nur durch einfache Kombination von Klassen aus den Quellen gebildet werden, wie dies in TSIMMIS erfolgt, sondern auch durch die Einbeziehung von Elementen der Domain Map.

Aufgrund der Graphstruktur der Domain Map und der darin »verankerten« Daten werden damit auch Graphoperationen, wie die Traversierung von Pfaden oder die Berechnung der transitiven Hülle sowie Inferenzregeln als Integrationsoperationen, genutzt. Die Einbeziehung von Domänenwissen ermöglicht somit eine *semantische* Integration von Daten, die wegen fehlender direkter Beziehungen ohne Ausnutzung dieses Wissens nicht kombiniert werden könnten. Allerdings ist dazu eine sorgfältige und oftmals aufwändige Modellierung des Domänenwissens notwendig.

6.3 Aufgaben und Probleme der Integration

Der Zugriff auf Daten aus mehreren Quellen erfordert Integrationstechniken auf verschiedenen Ebenen. Um den Zugriff durchführen zu können, werden verschiedene Konzepte des Zugriffs auf Datenbanken im Web sowie Middleware-Systeme benötigt, die innerhalb einer geeigneten Systemarchitektur die systembedingte Heterogenität überwinden. Auf allgemeine Konzepte und Architekturen sind wir im vorhergehenden Abschnitt eingegangen. Techniken des Zugriffs auf Datenbanken im Web werden in anderen Beiträgen in diesem Band vorgestellt (siehe etwa Kapitel 4). An dieser Stelle wollen wir uns daher zunächst auf die Probleme konzentrieren, die man im Wesentlichen unter dem Begriff semantische Heterogenität zusammenfassen kann.

Web-Anwendungen arbeiten in der Regel mit semistrukturierten Daten, so dass sich die Frage stellt, ob und inwieweit Integrationstechniken übertragen werden können, die ursprünglich für klassische Anwendungsszenarien entwickelt wurden. Mit klassischen Anwendungsszenarien sind hier Szenarien gemeint, in denen die Daten in den zu integrierenden Datenquellen in vollstrukturierter Form, also in Datenbanken mit einem Datenbankschema vorliegen. Eine solche Situation liegt z.B. dann vor, wenn in einem Unternehmen Daten aus zwei (oder mehreren) bestehenden Informationssystemen integriert werden sollen, die jeweils auf einem Datenbanksystem zur Verwaltung der Daten basieren. Die hierfür entwickelten Integrationsverfahren setzen voraus, dass die Modellierung der Daten in einem bekannten Datenmodell vorliegt (z.B. im ER-Modell, im relationalen Datenmodell oder im objektorientierten Datenmodell).

Unter gewissen Bedingungen lassen sich solche bestehenden Integrationstechniken einfach für semistrukturierte Daten übernehmen. Die Arbeit mit semistrukturierten Daten birgt allerdings zusätzliche Komplikationen, die in den meisten Fällen einer eigenen Behandlung bedürfen.

Die für klassische Anwendungsszenarien entwickelten Integrationsverfahren wie etwa [SpPD92] (für einen Überblick über verschiedene solche Techniken siehe z.B. [Conr97, PiBE95]) haben mehrere Voraussetzungen für ihre Anwendbarkeit – insbesondere die folgenden, die in der Regel nicht explizit postuliert werden:

- Die Datenbankschemata der einzelnen Quellen müssen bekannt sein. Das Gleiche gilt für ihre Semantik.
- Die Korrespondenzen zwischen Elementen der Schemata verschiedener Quellen müssen vollständig bekannt sein.
- Alle Integrationsverfahren basieren (in der einen oder anderen Weise) auf Korrespondenzen, die die semantischen Beziehungen zwischen Schemaelementen beschreiben. Dies setzt natürlich voraus, dass die Semantik der einzelnen Datenquellen bekannt ist (siehe erste Bedingung).

In der Praxis sind diese Voraussetzungen auch in klassischen Anwendungsbereichen nicht immer gegeben. Gerade bei Altsystemen (*legacy systems*) ist häufig die Semantik der Daten nicht vollständig bekannt. Bietet ein System nur verschiedene Prozeduren (und keine Anfragesprache zur Formulierung beliebiger Anfragen) als Schnittstelle zum Zugriff auf die von dem System verwalteten Daten an, so ist zudem auch die Struktur des Datenbankschemas nur teilweise bekannt bzw. zugänglich. Diese Probleme finden wir im Zusammenhang mit semistrukturierten Daten wieder, wobei sie für semistrukturierte Daten eher die Regel als die Ausnahme darstellen.

6.3.1 Datenmodelle und Sprachen

Datenbanken im Web können unterschiedliche Datenmodelle verwenden. Dies führt bei der Integration zu den in Abschnitt 6.3.2 näher erläuterten Datenmodellkonflikten. Einerseits können die Daten in Datenmodellen beschrieben sein, die von kommerziellen Datenbanksystemen unterstützt werden. In vielen Bereichen werden relationale Datenbanksysteme eingesetzt, so dass diese Systeme auch häufig als Datenquellen im Web angeboten werden. Daneben könnten sicherlich auch andere Systeme vorkommen, etwa objektorientierte Datenbanksysteme. Auf diese eher »traditionellen« Datenmodelle wollen wir hier nicht weiter eingehen und gehen davon aus, dass sie bekannt sind.

Handelt es sich um semistrukturierte Daten, die eine Quelle im Web anbietet, so sind auch hier verschiedene Datenmodelle denkbar, denen allerdings im Wesentlichen die gleichen Prinzipien zugrunde liegen. In [AbBS00] werden diese Prinzipien mit Hilfe von Graphen formalisiert (vgl. auch Kapitel 1).

Da sich praktisch alle Datenmodelle bzw. Formulierungsformalismen für semistrukturierte Daten auf eine solche Graphrepräsentation zurückführen lassen, ist der Vergleich (und damit gegebenenfalls auch die nachfolgende Integration) semistrukturierter Dokumente auf der Basis der allgemeinen Graphstruktur möglich. Da diese Struktur im Wesentlichen auch objektorientierten Datenmodellen zugrunde gelegt werden kann, ist eine objektorientierte Sicht auf semistruktu-

rierte Daten sehr einfach zu erhalten (vgl. auch [AbBS00, Kapitel 2]). Probleme bereitet dabei dann vor allem die (strenge) Typisierung, die die meisten kommerziell unterstützten objektorientierten Datenmodelle verlangen.

In Hinblick auf die Integration von Daten aus verschiedenen Quellen stellt sich die Frage, welches Datenmodell bzw. welcher Beschreibungsformalismus auf der globalen Ebene (z.B. in einem Mediator) am geeignetesten ist. In diesem Zusammenhang muss man sich auch über die Vorgehensweise zur Integration Gedanken machen. Prinzipiell kann man zwei Herangehensweisen unterscheiden (vgl. auch [Hass99]):

- Bottom-up-Integration: Ausgehend von den vorhandenen Daten und ihren Schemata in den Datenquellen (lokalen Datenbanken) werden Korrespondenzen zwischen den lokalen Schemata (bzw. Daten) bestimmt, mit deren Hilfe ein globales Schema aufgebaut werden kann. Hierbei gehen alle Schemainformationen, die die lokalen Systeme der Integration zur Verfügung stellen, in das integrierte globale Schema ein. Das so entstandene Schema stellt damit eine integrierende Sicht über alle zu integrierenden Datenbestände dar. Dies wird auch als *Global-as-View*-Ansatz bezeichnet [Hale01].
- Top-down-Integration: Um ein bestimmtes Anwendungsfeld möglichst optimal zu unterstützen, wird ein geeignetes globales Schema entworfen. In verschiedenen Anwendungsbereichen gibt es bereits Referenzschemata, die durch Standardisierungsgremien festgelegt wurden bzw. noch werden. Liegt damit das globale Schema fest, werden die lokalen Schemata auf dieses globale Schema abgebildet. Hierzu müssen in der Regel entsprechende Datenmodelltransformationen und gegebenenfalls notwendige Schema-Umstrukturierungen vorgenommen werden. Im Prinzip kann man also jedes lokale Schema als eine Sicht auf das vorgegebene globale Schema auffassen (*Local-as-View*).

Die Vorgehensweise bei der Integration bestimmt zum Teil auch die Wahl des globalen Datenmodells. Bei der Bottom-up-Integration bietet es sich an, das globale Datenmodell so zu wählen, dass die vorhandenen lokalen Schemata einfach in das globale Datenmodell transformiert werden können, so dass dann die eigentliche Integration in einem Datenmodell (dem globalen) durchgeführt werden kann. Bei der Top-down-Integration wird sehr häufig ein Referenzschema verwendet, das bereits in einem bestimmten Datenmodell formuliert ist. Damit ist das globale Datenmodell in der Regel bereits festgelegt. Da die Datenmodelle der lokalen Datenbanken auch feststehen, sind die notwendigen Transformationsschritte ebenfalls offensichtlich.

6.3.2 Integrationskonflikte bei semistrukturierten Daten

Für die Betrachtung der möglichen Integrationskonflikte verwenden wir eine einfache aber treffende Unterscheidung entsprechend den Ursachen, denen praktisch

alle auftretenden Integrationskonflikte zugeordnet werden können. Die Ursachen liegen auf drei verschiedenen Ebenen:

- Datenmodellheterogenität,
- Schemaheterogenität (oder heterogene Modellierung),
- Heterogenität auf der Datenebene.

Eine andere weit verbreitete Klassifikation von Konfliktarten wurde in [SpPD92] eingeführt. Dort werden extensionale Konflikte (in [SpPD92] als *semantic conflicts* bezeichnet), Beschreibungskonflikte, Heterogenitätskonflikte sowie strukturelle Konflikte unterschieden. Während die Heterogenitätskonflikte genau Datenmodellkonflikten entsprechen, strukturelle Konflikte eine spezielle Form von Schemakonflikten darstellen, umfassen die anderen beiden Konfliktarten aus [SpPD92] jeweils sowohl bestimmte Schemakonflikte als auch Datenkonflikte. Auf der anderen Seite werden durch die Klassifikation in [SpPD92] jedoch nicht alle Datenkonflikte erfasst. Für unsere Zwecke ist daher die Diskussion der Integrationskonflikte entsprechend der oben angegebenen Unterscheidung besser geeignet.

Konflikte durch Datenmodellheterogenität

Hiermit bezeichnet man im Allgemeinen die Probleme, die durch die Verwendung unterschiedlicher Datenmodelle zur Darstellung von Datenbankschemata entstehen. Verschiedene Datenmodelle bieten oft unterschiedliche Mengen von Modellierungskonstrukten an und können sich damit hinsichtlich ihrer Ausdrucksmächtigkeit deutlich unterscheiden. In der Regel versucht man, durch eine Transformation in ein gemeinsames Datenmodell (hier wird meist gefordert, dass dieses Datenmodell hinsichtlich der Ausdrucksfähigkeit eine Obermenge darstellen muss) die Probleme der unterschiedlichen Repräsentation zu überwinden.

Abhängig von den vorliegenden Datenmodellen und dem gewählten gemeinsamen Datenmodell können aber bei der Transformation semantische Informationen verloren gehen (wenn sie im gemeinsamen Datenmodell nicht oder nicht adäquat repräsentierbar sind).

Bei der Integration semistrukturierter Daten mit (voll-)strukturierten Daten, z.B. aus einer relationalen Datenbank, stellt sich die Frage nach der Wahl des gemeinsamen (globalen) Datenmodells, in dem das Integrationsergebnis dargestellt werden soll.

- Wird als Zieldatenmodell ein (voll-)strukturiertes Modell wie etwa das relationale Datenmodell angestrebt, so müssen die semistrukturierten Daten in das (voll-)strukturierte Modell überführt werden. Hierfür ist eine geeignete Transformation zu finden, die aus dem semistrukturierten Dokument die zusammengehörigen Daten entsprechend zusammenstellt. Eine solche Transformation kann etwa auf XSLT basieren und/oder Anfragesprachen für semistrukturierte Daten (siehe Kapitel 3) nutzen.

- Sollen die integrierten Daten wiederum in Form semistrukturierter Dokumente zur Verfügung gestellt werden, müssen die (voll-)strukturierten Daten in die entsprechende semistrukturierte Repräsentation überführt werden. Die Transformationsrichtung dürfte in der Praxis die einfachere sein.

Selbst wenn semistrukturierte Daten »nur« mit anderen semistrukturierten Daten integriert werden sollen, können verschiedene Repräsentationsformen für semistrukturierte Daten vorliegen, so dass auch hier die Festlegung des Zieldatenmodells (z.B. in Form eines allgemeinen Graphmodells) erforderlich wird und entsprechende Transformationen zwischen den vorliegenden Datenmodellen gefunden werden müssen.

Konflikte durch Schemaheterogenität

Auch wenn die gleichen Mengen von Real-Welt-Objekten in zwei (oder mehreren) Schemata modelliert sind, werden sie doch oft mit unterschiedlichen Eigenschaften repräsentiert, da die Anwendungen, die die Datenbestände der unterschiedlichen Quellen nutzen, verschiedene Anforderungen stellen, so dass nicht überall die gleichen Eigenschaften zum selben Real-Welt-Objekt gespeichert sein müssen. Durch die Integration ergänzen sich diese Daten gegenseitig. Andererseits erschwert dies aber auch die Erkennung, welche Schemaelemente einander entsprechen können (z.B. beim Schema Matching).

Eine andere, häufig vorkommende Form von Integrationskonflikten auf Schemaebene sind Homonyme und Synonyme bei den Bezeichnern von Mengen (Klassen, Relationen) und Eigenschaften (Attributen). Bei semistrukturierten Daten können diese z.B. als homonyme und synonyme Tags (Elementbezeichner) in XML-Dokumenten auftreten. Das Entdecken solcher Homonyme und Synonyme ist nur bedingt automatisierbar. Im Zusammenhang mit anderen Techniken (wie etwa Schema Matching oder durch Analyse der konkret vorliegenden Datenbestände) können Kandidatenpaare für Homonyme und Synonyme gefunden werden, die durch einen Experten überprüft werden müssen.

Fast immer ist es auch möglich, denselben Sachverhalt in unterschiedlichen Strukturierungen innerhalb eines Datenmodells auszudrücken. Dies gilt natürlich auch für semistrukturierte Daten.

So lassen sich z.B. in XML drei zusammengehörige Elemente A, B und C, zu denen jeweils ein Wert vorhanden ist (bzw. sein kann), in verschiedenen Konstellationen zusammenstellen. Einige dieser Möglichkeiten der strukturellen Anordnung sind in Abb. 6-4 exemplarisch als Baumstruktur angegeben (X, Y und Z sind zusätzliche Elemente/Tags zur Zusammenfassung von Elementen).

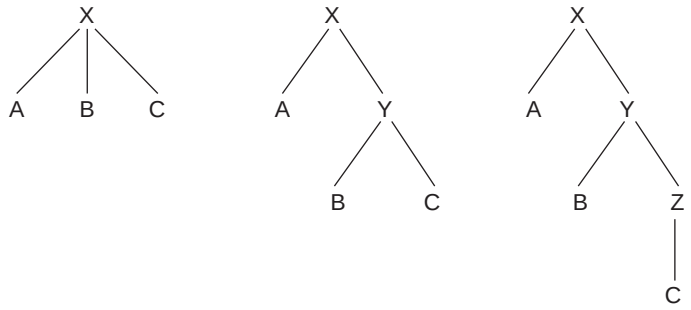


Abb. 6-4: *Verschiedene Dokumentstrukturen mit gleichem Inhalt*

Wird ferner noch die Reihenfolge der Elemente berücksichtigt, entstehen aufgrund der möglichen Permutationen noch viel mehr Varianten.

Ein weiteres strukturelles Problem lässt sich im Prinzip auf eine Entwurfsfrage zurückführen: Wann wird etwas als Wert eines Elements und wann als Wert eines Attributs zu einem Element repräsentiert? Bei der Integration von Daten aus verschiedenen Quellen können beide Formen aufeinander stoßen.

Beispiel. In zwei semistrukturierten Dokumenten `Datei 1` und `Datei 2` finden sich die in Abb. 6-5 dargestellten Einträge.

```
<!-- Datei 1 -->
...
 <name email="jo@test.de">
   Hamann
 </name>
 <vorname>
   Joachim
 </vorname>
 ...
```

```
<!-- Datei 2 -->
...
 <name>
  <nachname>Hamann</nachname>
  <vorname>Joachim</vorname>
  <email>jo@test.de</email>
 </name>
...
```

Abb. 6-5: *Daten-/Metadaten-Konflikt bei Verwendung von Attributen*

Auch wenn beide Dokumente dieselbe Information enthalten, liegt sie doch in verschiedenen Strukturen vor. In `Datei 1` ist die E-Mail-Adresse als Attributwert zu dem Element `name` zu finden, wobei `name` den Nachnamen einer Person bezeichnet. In `Datei 2` ist hingegen die E-Mail-Adresse als eigenständiges Element zu finden, das zusammen mit den Elementen `nachname` und `vorname` zu einem Element `name` zusammengefasst ist. □

Eine weitere, immer wieder anzutreffende Strukturierungsform stellt so genannten *mixed content* dar, bei dem Daten (PCDATA) und Elemente gemischt in anderen Elementen vorkommen können. Eine Mixed-content-Form ist z.B.:

```
<name>Hamann
  <vorname>Joachim</vorname>
</name>
```

Solche Mixed-content-Formen kommen gerade bei Textdokumenten vor, in denen in Textteilen etwa einzelne Wörter mit bestimmten Merkmalen ausgezeichnet werden. In

```
<text>Dies ist ein <em>wichtiger</em> Text</text>
```

sehen wir z.B. die Hervorhebung des Wortes `wichtiger` durch eine Markierung, die bedeuten soll, dass dieses Wort durch kursive Schrift beim Ausdruck hervorgehoben werden soll (wenn man die aus HTML bekannte Bedeutung von `<em>...</em>` zugrunde legt).

Eine besondere Konfliktart auf der Schema- oder Modellierungsebene stellen *extensionale Konflikte* dar. Diese beziehen sich darauf, dass die Mengen einander entsprechender Datenobjekte (also die Extensionen von Klassen oder Relationen) in unterschiedlichen Beziehungen zueinander stehen können:

- Sie können die gleiche Menge an Real-Welt-Objekten repräsentieren,
- die Mengen der repräsentierten Real-Welt-Objekte können in einer Teilmengenbeziehung zu einander stehen,
- sie können sich überlappen oder
- sie sind disjunkt zueinander.

Bevor man extensionale Konflikte erkennen und behandeln kann, müssen zuerst die Schemaelemente der zu integrierenden Datenbankschemata bestimmt werden, die semantisch gleiche Dinge repräsentieren. Hierbei kann *Schema Matching* [RaBe01] helfen, womit mögliche Übereinstimmungen zwischen Schemata gefunden werden können. Dies führt zu potenziellen Inter-Schema-Korrespondenzen, die nachfolgend überprüft werden müssen. Neben Expertenwissen kann man dann auch Analyseverfahren wie *Record Linkage* [FeSu69, ElVE02] auf vorhandene Datenbestände anwenden um festzustellen, welche extensionale Beziehung konkret vorliegt.

Während in »klassischen« Datenbanken relativ klar ist, welche Schemaelemente Mengen von (Daten-) Objekten repräsentieren, ist dies bezüglich semistrukturierter Daten nicht ganz so einfach. Zunächst einmal lassen sich beispielsweise ganze Mengen von XML-Dokumenten miteinander vergleichen. Andererseits können auch innerhalb von XML-Dokumenten Mengen von Teildokumenten z.B. als Elemente eines gemeinsamen Elternelements auftreten. Bezogen auf die zugrunde liegenden DTDs muss man also zuerst einmal die Frage beantworten, welche Elemente in einer DTD Mengen von Objekten darstellen.

Konflikte durch Heterogenität auf Datenebene

Auf der Datenebene können vielfältige Integrationsprobleme auftreten, z.B.:

- Die Datenquellen selbst können bereits fehlerhafte Dateneinträge enthalten (z.B. aufgrund von Tippfehlern). Um bei der Integration zu erkennen, dass im Prinzip der gleiche Wert aus zwei verschiedenen Quellen geliefert wird, können »kleinere« Fehler (wie einfache Tippfehler) durch Ähnlichkeitsanalysen

(z.B. Vergleich der Repräsentationen in einem phonetischen Code) behoben werden.

- Es können unterschiedliche Repräsentationen für die gleiche Information verwendet werden. Da Daten (in XML) innerhalb von Elementen immer als Zeichenketten dargestellt werden, lassen sich oft verschiedene Zeichenketten als Repräsentation für dieselbe Information angeben: So kann die Zeichenkette »1« ebenso wie die Zeichenkette »sehr gut« die gleiche Bewertung (Note) repräsentieren. Zusätzlich können in verschiedenen XML-Dokumenten unterschiedliche Zeichensätze zur Darstellung verwendet werden. Wird dies im Header des XML-Dokuments angegeben, ist eine Konvertierung problemlos möglich.
- Die Identifizierung von (Daten-) Objekten stellt ein sehr großes Problem in semistrukturierten Dokumenten bzw. Datenquellen dar. Durch die Verwendung von ID- und IDREF-Attributen lassen sich innerhalb eines Dokuments Identitäten einführen und zur Grundlage von Beziehungen (Referenzen) machen. Referenzierungen über Dokumentgrenzen hinweg sind beispielsweise unter Verwendung von XLink und XPointer zu realisieren. Allerdings haben hier die Anwendungen, die diese Dokumente erzeugen bzw. verändern, sicherzustellen, dass diese Mechanismen richtig verwendet werden.

Bei der Integration semistrukturierter Dokumente aus verschiedenen Quellen nützen diese Mechanismen nur wenig. Die ID- und IDREF-Attribute liefern nur eine eindeutige Identität innerhalb eines Dokuments. Für einen Vergleich mit Elementen eines anderen Dokumentes sind sie nicht zu verwenden. Liegen Referenzen auf der Basis von XLink bzw. XPointer zwischen den zu integrierenden Dokumenten vor, ist ein (evtl. sogar größerer) Teil der Integrationsarbeit schon geleistet. Ist dies nicht der Fall, steht man vor demselben Problem, zusammengehörige Elemente (die dasselbe beschreiben bzw. die zueinander in Beziehung stehen) bei der Integration zu bestimmen, wie sie bereits bei der »klassischen« Datenbankintegration vorkommen (vgl. [Conr97: Kap. 7.3]). Es können hier sowohl wertbasierte Verfahren zur Bestimmung von Übereinstimmungen als auch – für semistrukturierte Daten zusätzlich interessant – strukturbasierte Verfahren zum Einsatz kommen.

Diese Beispiele für Integrationsprobleme auf der Datenebene veranschaulichen die Komplexität der zu behandelnden Probleme. Eine ausführlichere Darstellung dieser und weiterer Probleme sowie Ansätze zur Behandlung findet man z.B. in [HeSt98, RaDo00].

6.3.3 Integrierbarkeit semistrukturierter Daten

Im Folgenden diskutieren wir verschiedene typische Situationen, die bei der Integration semistrukturierter Daten (mit anderen semistrukturierten Daten oder (voll-) strukturierten Daten aus klassischen Datenbanken) auftreten können. Der Einfachheit halber nehmen wir an, dass die zu integrierenden semistrukturierten Daten in Form von XML-Dokumenten vorliegen. Diese Annahme schränkt die

Allgemeingültigkeit der Diskussion auch für andere Repräsentationsformen semistrukturierter Daten in keinster Weise ein.

- Es sind XML-Dokumente mit der zugrunde liegenden DTD gegeben:
  Dieses stellt den einfachsten Fall dar, da er im Wesentlichen der »klassischen Situation« entspricht, für die bestehende Integrationsverfahren entwickelt wurden. Somit treten hier dieselben Probleme wie für entsprechende nicht semistrukturierte Daten mit einem gegebenen Datenbankschema auf. Eine DTD gibt zunächst einmal nur eine Struktur an, deren Semantik damit noch nicht bekannt ist (dies ist vergleichbar mit so genannten semantisch armen Datenmodellen, die nur wenige Modellierungskonzepte unterstützen). Handelt es sich aber um eine anwendungsbereichsspezifische DTD, ist die Situation bereits deutlich einfacher, da hier die Semantik der DTD typischerweise von Standardisierungsgremien festgelegt wurde (wie etwa die XML-basierte Variante des EDIFACT-Standards, die im Bereich des B2B–E-Commerce an Bedeutung gewinnt). Auch solche Standards bieten oft noch keine perfekte Lösung, da sie erfahrungsgemäß verschiedene Details nicht explizit festlegen, wodurch es durch unterschiedliche Auslegungen und Realisierungen der Software-Hersteller zu Integrationskonflikten kommen kann.
- Es sind XML-Dokumente ohne die zugrunde liegende DTD gegeben:
  Die fehlende DTD kann zum Teil durch Extraktionstechniken rekonstruiert werden, die die Struktur der vorhandenen XML-Dokumente analysieren und eine Grammatik bzw. DTD ableiten, der alle vorhandenen Dokumente genügen. In [AhMN94] wird beispielsweise ein Verfahren vorgeschlagen, das die vorhandenen Dokumente hinsichtlich ihrer Struktur analysiert und dabei sukzessiv einen Automaten aufbaut, der diese (und ähnlich strukturierte) Dokumente erkennt. Aus der Konstruktion des Automaten kann dann eine Grammatik (resp. DTD) abgeleitet werden. Eine auf diese Weise bestimmte DTD kann natürlich nur die Struktur der vorhandenen und nicht die aller aus Anwendungssicht erlaubten XML-Dokumente erfassen und ist somit prinzipiell als unvollständig zu betrachten. Dies bedeutet, dass später hinzugefügte XML-Dokumente dieser DTD nicht immer entsprechen müssen. Selbstverständlich ist mit der abgeleiteten DTD die Semantik der XML-Dokumente noch nicht bekannt. Die Bedeutung der verwendeten Tags sowie deren (semantischen) Abhängigkeiten bleiben zunächst offen und können nur durch Anwendungsexperten bestimmt werden. Der Einsatz solcher Verfahren setzt allerdings voraus, dass hinreichend viele Dokumente des zu bestimmenden Typs verfügbar sind (und aus der Datenquelle extrahiert werden können), um einen möglichst guten Automaten (hinsichtlich der korrekten Erkennung von Dokumenten) zu erhalten.

Wird für die XML-Dokumente einer Datenquelle keine oder nur eine unvollständige DTD von der Quelle bereitgestellt oder extrahiert, kann man versuchen, weitere strukturelle oder semantische Information zur Vervollständigung der

DTD durch den Einsatz von Data-Mining-Verfahren aus den vorhandenen Daten zu gewinnen. So können etwa Verfahren zur Extraktion von Assoziationsregeln genutzt werden, um funktionale und andere Arten von (Daten-) Abhängigkeiten zu finden. Die prinzipielle Einsatzmöglichkeit verschiedener Arten von Data-Mining-Verfahren für diese Zwecke wird z.B. in [HöCo98] diskutiert.

Da Data-Mining-Verfahren keine perfekten Ergebnisse liefern können – in dem Sinne, dass die Gültigkeit z.B. einer Assoziationsregel bereits als Ergebnis mit einer Unsicherheit (Wahrscheinlichkeit bzw. Support- und Confidence-Werte) versehen ist –, ist es notwendig, diese Unsicherheit im Rahmen der Integration mit zu berücksichtigen. In [AlCo01] werden dazu die verschiedenen Ursachen der Unsicherheit (die oben bereits erwähnt wurden) sowie exemplarisch die notwendigen Erweiterungen bestehender Integrationstechniken, z.B. um mit Wahrscheinlichkeiten behaftete Schemainformationen oder Korrespondenzen, diskutiert.

6.4 Anfragebearbeitung

Die wesentliche Aufgabe eines Mediators ist die Verarbeitung von Anfragen, die auf Basis des globalen Schemas formuliert wurden. Hierzu müssen diese in Teilanfragen zerlegt werden, über die entsprechenden Wrapper den einzelnen Quellen zur Bearbeitung übergeben und die zurückgelieferten Ergebnisse schließlich entsprechend der globalen Schemadefinition kombiniert werden. Obwohl dieses Vorgehen prinzipiell mit der Anfragebearbeitung in verteilten Datenbanksystemen vergleichbar ist, bestehen bei der Anfragebearbeitung in Web-Integrationssystemen doch einige spezielle Probleme, die besondere Techniken erfordern:

- So ist eine vergleichsweise große Anzahl von Quellen zu berücksichtigen, die auch noch räumlich weit verteilt sein können und weitgehend autonom arbeiten.
- Weiterhin unterstützen die einzelnen Quellsysteme oft nur eingeschränkte Anfragemöglichkeiten (z.B. im Fall von Web-Formularen als Anfrageschnittstelle nur einfache Selektionen mit wenigen Prädikaten über bestimmte Attribute) und stellen darüber hinaus kaum Kosteninformationen bereit.
- Schließlich führt auch die Charakteristik des Web und der verwendeten Protokolle zu nur schwer abzuschätzenden Antwortzeiten.

In den folgenden Abschnitten werden wir daher einige ausgewählte Techniken für Mediatorsysteme vorstellen, die diesen speziellen Problemen gewidmet sind. Wir setzen dabei Grundkenntnisse der Anfragebearbeitung voraus, die von einschlägigen Lehrbüchern [SaHe99, HäRa01, YuMe98] und Übersichtsartikeln [Grae93, Koss00] vermittelt werden. Weiterhin gehen wir im Wesentlichen von relationalen Daten aus, weil die meisten der betrachteten Verfahren in diesem Kontext entwickelt wurden. Eine Übernahme auf andere Datenmodelle inklusive XML ist in den meisten Fällen jedoch grundsätzlich möglich.

6.4.1 Grundprinzipien der Anfragebearbeitung

Der Prozess der Anfragebearbeitung in Datenbanksystemen wird üblicherweise in mehrere Schritte aufgeteilt (Abb. 6-6).

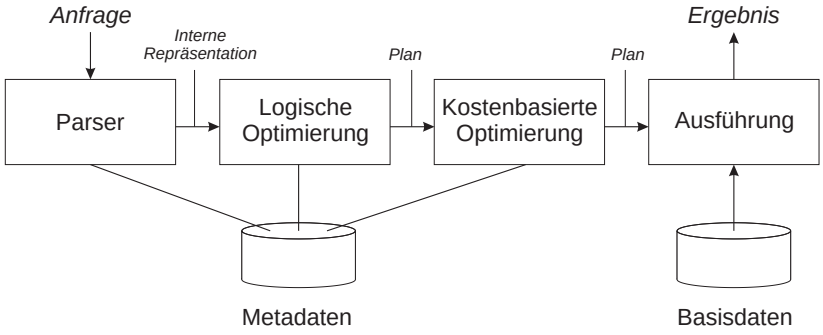


Abb. 6-6: *Prozess der Anfragebearbeitung [HFLP89]*

Eine Anfrage, formuliert in einer deklarativen Sprache wie SQL oder XQuery, wird zunächst vom Parser auf syntaktische Korrektheit überprüft und in eine interne Repräsentation – meist ein Anfragebaum mit Operationen wie Verbund, Selektion, Relationenzugriff usw. als Knoten – überführt. Im nächsten Schritt erfolgt eine logische Optimierung (auch als *Rewriting* bezeichnet), bei der die Anfrage unter Verwendung von Transformationsregeln noch ohne Berücksichtigung physischer Aspekte wie Relationengröße etc. umgeformt wird. Dabei werden beispielsweise Sichten aufgelöst oder redundante Teilausdrücke entfernt.

Der dritte Schritt ist die kostenbasierte Optimierung. Hier werden zu einem Anfragebaum verschiedene Ausführungspläne generiert, indem für die logischen Operationen physische Operatoren (d.h. Implementierungen für Operationen) eingesetzt und die Reihenfolge für die Berechnung von Verbunden festgelegt werden. Kommerzielle DBMS verwenden zur Generierung der einzelnen Pläne und zum effizienten Einschränken und Durchsuchen des Lösungsraums meist eine Technik auf Basis dynamischer Programmierung.

In Mediatorsystemen umfasst dieser Schritt auch die Dekomposition in Teilanfragen, die an die einzelnen Quellen zur Bearbeitung weitergeleitet werden. Der Anfrageoptimierer muss dazu die relevanten Quellen identifizieren und festlegen, welche Operationen von welchem Quellsystem ausgeführt werden sollen. Verfügen einzelne Quellen nur über eingeschränkte Anfragefunktionalität, muss dies vom Optimierer ebenfalls berücksichtigt werden, indem eventuell zusätzliche Operationen im Mediator ausgeführt werden.

Aus der Menge der möglichen Pläne für eine Anfrage wird schließlich der Plan mit den geringsten Kosten ausgewählt und der Ausführungskomponente zur Abarbeitung übergeben.

6.4.2 Beschreibung von Quellen

In Szenarien mit einer Vielzahl von Quellen mit teilweise redundanten oder partiellen Informationen kommt der Identifikation der für die Bearbeitung einer Anfrage relevanten Quellen eine große Bedeutung zu: einerseits, um die Zahl der während der Optimierung zu untersuchenden Pläne einzuschränken, und andererseits natürlich im Interesse einer effizienten Ausführung. So kann zum Beispiel bei Quellen mit redundanten Inhalten die Quelle mit den geringsten Zugriffskosten zur Bearbeitung ausgewählt werden, oder es lassen sich Quellen ausschließen, die nur nicht benötigte Fragmente der Gesamtrelation beinhalten. Allerdings gibt es auch Fälle, in denen ein solcher Ausschluss nicht möglich ist: Verwalten etwa mehrere Quellen Informationen über die gleichen Produkte, jedoch mit verschiedenen Preisen, so müssen bei einer Anfrage nach dem günstigsten Preis alle Quellen einbezogen werden.

Grundsätzlich wird natürlich bei der Registrierung einer Quelle beim Mediator angegeben, welche Relationen die betreffende Quelle exportiert. Zur Nutzung weiterer Optimierungen sind jedoch noch die Inhalte der Quellen in geeigneter Form zu beschreiben und dem Mediator als Metadaten zur Verfügung zu stellen. In Systemen, die dem Local-as-View-Ansatz folgen, wird dies durch die Definition der Quellrelationen als Sicht über die globalen Relationen erreicht. Selektionen und Projektionen als Teil der Sichtdefinition liefern hierbei die gewünschte Einschränkung. So lässt sich beispielsweise eine Quelle, die zu einer globalen Relation `filme(titel, genre, jahr)` nur SF-Filme seit 1995 anbietet, durch eine konjunktive Anfrage der folgenden Form beschreiben [LeRO96]:

$$\text{sf-filme(f)} \subseteq \text{filme(c), genre = 'SF', jahr} \geq 1995$$

Das Ableiten eines Anfrageplans unter Verwendung solcher Quellenbeschreibungen lässt sich auf das Problem der Anfrageauswertung auf Sichten zurückführen (Query-Containment-Problem), das für den allgemeinen Fall konjunktiver Anfragen als NP-vollständig bekannt und damit sehr aufwändig ist.

Eine andere, vereinfachte Lösung basiert auf dem aus dem Bereich verteilter Datenbanken bekannten Ansatz der qualifizierten Relationen [Dada96]. Hierbei wird zur Quellrelation ein Selektionsprädikat angegeben, welches den Inhalt bezüglich der globalen Relation einschränkt, also z.B. »jahr $\geq$ 1995«. Für eine Anfrage nach Filmen der Jahre bis 1993 kann aufgrund der Tatsache, dass der Ausdruck »jahr $\leq$ 1993 $\wedge$ jahr $\geq$ 1995« nie erfüllt wird, die betreffende Quelle für die Bearbeitung der Anfrage ausgeschlossen werden.

Ein weiterer wichtiger Aspekt der Beschreibung von Quellen ist die Anfragefähigkeit. Hier sind grundsätzlich zwei verschiedene Vorgehensweisen möglich:

- Anfrageeinschränkungen oder auch »negative« Fähigkeiten in Form einer vorgegebenen Menge von parametrisierbaren Anfrageschablonen,
- »positive« Fähigkeiten zur Angabe der Operationen, die von der Quelle unterstützt werden, sowie eventueller Kombinationen davon.

Die Technik der Anfrageschablonen wurde erstmals im Rahmen des TSIMMIS-Projektes eingesetzt. Dort werden Wrapper anhand einer Spezifikation zur Anfrageübersetzung generiert [PGGU95]. Eine solche Spezifikation umfasst eine Menge von MSL-Regeln (siehe auch Abschnitt 6.2), die jeweils aus einer Anfrageschablone – formuliert in der globalen Anfragesprache – sowie einer Aktion als Folge von Operationen zur Umsetzung dieser Anfrage in Aufrufe der Quelle besteht. Anfrageschablonen beinhalten Platzhalter (notiert als $1, $2, etc.), die beim konkreten Aufruf mit Werten belegt werden. So kann zum Beispiel eine Quelle, die nur Selektionen nach dem Titel oder dem Jahr unterstützt, durch die beiden folgenden Schablonen beschrieben werden:

```
<filme F> :- <archiv { F:<film {<titel $1>}> }> @source
<filme F> :- <archiv { F:<film {<jahr $2>}> }> @source
```

Die Definition in Form von logischen Regeln ermöglicht dabei auch die Ausführung von Anfragen, die nicht direkt mit der Schablone übereinstimmen, sondern logisch äquivalent sind bzw. weitere Bedingungen umfassen. Anfragen der letzteren Form können durch zusätzliche Filter im Wrapper unterstützt werden.

Ein anderer Ansatz für Anfrageeinschränkungen wird u.a. im FraQL-System [EHSS00] angewendet. Hier werden so genannte Query Constraints angeboten, die zu den Relationen einer Quelle die Selektionsattribute einschließlich der unterstützten Operatoren (<, >, = etc.) sowie der Kombinationen daraus spezifizieren, die von der Quelle unterstützt werden. Das obige Beispiel kann damit wie folgt formuliert werden:

```
alter view sf_filme set query constraints (
predicates ((titel, =), (jahr, <))
combinations ((titel), (jahr))
)
```

Im Rahmen der Anfragedekomposition wird vom Optimierer versucht, Selektionen so zu zerlegen, dass diese Constraints erfüllt werden. Gelingt dies nicht, wie im Fall der Anfrage `select * from sf_filme`, wird die Anfrage mit einer Fehlermeldung zurückgewiesen.

Die Idee der Beschreibung positiver Fähigkeiten kommt u.a. im Garlic-System [HKWY97] von IBM zum Einsatz. Die zur Kommunikation mit der Quelle verwendeten Wrapper bieten hierbei so genannte Planungsfunktionen an, die vom Optimierer im Rahmen der Plangenerierung aufgerufen werden. Im Wesentlichen handelt es sich dabei um Funktionen zum Zugriff auf Relationen und zur Berechnung eines Verbundes, für die im Wrapper geeignete Ersetzungen (z.B. durch Aufruf eines CGI-Programms für eine Web-Anfrage) definiert sind. Solche Planungsfunktionen lassen sich mit Bedingungen versehen und implementieren somit Regeln für Teilpläne, die später vom Wrapper ausgeführt werden.

Bei der Generierung eines Plans versucht der Optimierer für den Zugriff auf eine Quellrelation eine Planungsfunktion des entsprechenden Wrappers zu verwenden, deren Bedingung erfüllt ist. Wird vom Wrapper beispielsweise keine Ver-

bundoperation unterstützt, so ist die entsprechende Planungsfunktion nicht definiert und kann vom Optimierer auch nicht aufgerufen werden. Auf diese Weise erlauben es Bedingungen auch, Einschränkungen der Anfragefähigkeiten der Quelle zu formulieren.

6.4.3 Anfrageoptimierung

In Abschnitt 6.4.1 haben wir bereits die wesentlichen Aufgaben und Schritte der Anfrageoptimierung skizziert: Zu einer Anfrage ist ein möglichst kostenoptimaler Plan zu finden, der eine exakte Beschreibung zur Ausführung der Anfrage (d.h. welche Operationen in welcher Reihenfolge) darstellt. Neben der Anwendung von Heuristiken, wie das möglichst frühzeitige Ausführen von Selektionen bzw. deren Ausführung durch die Quellsysteme, bilden im Kontext von Mediatorsystemen die Bestimmung der Reihenfolge von Verbundberechnungen sowie die Auswahl der konkreten Verbundmethode einen wesentlichen Schwerpunkt. Die Verbundreihenfolge beeinflusst nicht nur die Ausführungskosten im Sinne des Aufwandes (Ein-/Ausgabekosten, CPU-Zeit), sondern auch die Antwortzeit, d.h. die Zeit bis zum Eintreffen der ersten Ergebnisse. Gerade bei interaktiven Anfragen ist Letzteres von großer Bedeutung.

Optimierer in traditionellen Datenbanksystemen bevorzugen Verbundreihenfolgen in Form von linksorientierten Bäumen (left-deep trees). Diese liefern in Verbindung mit Sort-Merge- oder Hash-Join-Implementierungen bezüglich der Kosten bessere Ergebnisse. Diese Art von Bäumen ist durch den Zugriff auf mindestens eine Basisrelation (bzw. im Fall eines Mediators auf Relationen des Quellsystems) in jeder Verbundoperation gekennzeichnet (Abb. 6-7). Allerdings ergeben sich daraus längere Antwortzeiten, da aufgrund der unter Umständen notwendigen Sortieroperation für einen Sort-Merge-Join bzw. den Aufbau einer Hashtabelle für den Hash-Join zunächst die gesamte Basisrelation verarbeitet werden muss, bevor der eigentliche Verbund ausgeführt werden kann. Außerdem wird auf diese Weise auch eine parallele Ausführung der Teilpläne verhindert.

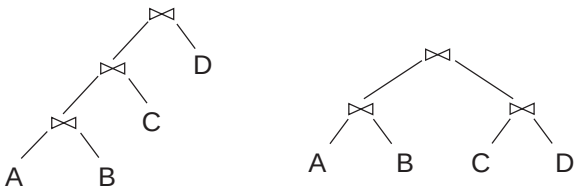


Abb. 6-7: *Left-deep-Tree vs. Bushy Tree*

Für Mediatorsysteme ist daher ein Ansatz besser geeignet, der auch bei parallelen Datenbanksystemen eingesetzt wird und bei dem man versucht, die Verbunde in Form »buschiger« Bäume (*bushy trees*) anzuordnen [DuSD95]. Im Beispiel in Abb. 6-7 ist es demzufolge möglich, die Teilpläne parallel auszuführen, d.h., die Anfragen an die Quellsysteme gleichzeitig abzuschicken. Obwohl dieser Plan hin-

sichtlich der Kosten eventuell ungünstiger ist – das Ergebnis von C⋈D muss als temporäre Relation materialisiert werden – ergibt sich eine im Vergleich zum linken Plan verbesserte Antwortzeit. Diese entspricht dem Maximum der Zeiten für beide Teilpläne plus der Zeit für den abschließenden Verbund, während beim ersten Plan die Gesamtzeit aus der Summe der Zeiten für die einzelnen Verbunde berechnet wird. Insbesondere wenn Verbunde auch von den Quellsystemen ausgeführt werden können, besitzt dieser Ansatz deutliche Vorteile. Weitere Verbesserungen lassen sich schließlich auch noch durch spezielle Implementierungen für Verbundoperationen erzielen (siehe Abschnitt 6.4.4).

Bei der Entscheidung für eine konkrete Verbundmethode spielen jedoch nicht nur Kostenaspekte eine wichtige Rolle, sondern es sind auch wieder die Fähigkeiten der Quellen zu berücksichtigen. Soll beispielsweise der Verbund zwischen zwei Relationen A und B aus verschiedenen Quellen gebildet werden, wobei die Quelle von B nur Anfragen mit einer Selektionsbedingung (wie z.B. `titel = «Matrix«`) unterstützt, so können traditionelle Verbundmethoden nicht eingesetzt werden. Stattdessen wird hier eine Methode benötigt, die auch als Bind-Join bezeichnet wird. Hierbei werden die Tupelwerte der äußeren Relation genutzt, um eine Selektion auf der inneren Relation auszuführen. Die gefundenen Tupel können dann direkt mit dem Tupel der äußeren Relation verknüpft werden.

Unabhängig davon, ob Verarbeitungskosten oder Antwortzeiten das entscheidende Kriterium für die Auswahl des optimalen Plans darstellen, müssen in jedem Fall Kostenschätzungen vorgenommen werden. Diese basieren auf

- Kostenfunktionen zur Bestimmung des Aufwands der verschiedenen Operationen inklusive Transferkosten, Kommunikationskosten etc.
- sowie Informationen über die beteiligten Relationen (z.B. Anzahl der Tupel, Größe eines Tupels, Werteverteilung).

Die Gesamtkosten eines Anfrageplans entsprechen den kumulativen Kosten aller Operationen des Plans. Da sich die Kosten jedes Teilbaums des Plans aus den Kosten der Operation an der jeweiligen Wurzel und den Kosten der Subpläne zusammensetzen, können die Gesamtkosten an der Wurzel des Plans abgelesen werden.

Die Kosten pro Operation ($Cost_{op}$) berechnen sich wiederum aus einem konstanten Aufwand für die Initialisierung ($Cost_{init}$) sowie dem Aufwand für das Lesen und die Verarbeitung des nächsten Tupels ($Cost_{next}$), der entsprechend mit der Kardinalität der Eingaberelation (plus eine Operation für den Test auf Ende der Relation) multipliziert werden muss:

$$Cost_{op} = Cost_{init} + Cost_{next} * (Card + 1)$$

Der Verarbeitungsaufwand umfasst dabei neben CPU- und E/A-Aufwand auch gegebenenfalls Transferkosten, die bei einem verteilten System einen beträchtlichen Anteil einnehmen können.

Die Kardinalität der Eingaberelation entspricht entweder der Größe der Ergebnisrelation des Subplans der jeweiligen Operation oder kann – für Basis-

relationen und – sofern überhaupt vorhanden – anhand von Statistiken zur betreffenden Relation direkt bestimmt werden. In jedem Fall wird die Kardinalität der Ergebnisrelation der jeweiligen Operation mit Hilfe des Selektivitätsfaktors *SF*, d.h. dem erwarteten Anteil der Tupel, die z.B. das Prädikat einer Selektion oder einer Verbundbedingung erfüllen, berechnet:

$Card_{out} = Card_{in} * SF$ (für Selektion)
$Card_{out} = Card_{in1} * Card_{in2} * SF$ (Verbund)

Der Selektivitätsberechnung liegen Annahmen über die aktuelle Werteverteilung der Attribute zugrunde, die beispielsweise aus Histogrammen gewonnen werden können. Sind diese nicht verfügbar, muss eine Standardverteilung der Datenwerte angenommen werden.

In Mediatorsystemen müssen die Kosteninformationen zu den Operationen, die in den Quellsystemen bzw. im Wrapper ausgeführt werden, sowie die Statistikinformationen zu den Quelldaten vom jeweiligen Wrapper in Form eines Kostenmodells bereitgestellt werden. Gerade für heterogene Web-Quellen ist dies aber oft sehr schwierig, da meist keine Informationen über physische Aspekte der Quellen vorliegen. Grundsätzlich sind zur Kostenbestimmung solcher Quellen die folgenden Ansätze möglich [Koss00]:

- individuelle Kostenmodelle für Wrapper,
- generisches Kostenmodell,
- adaptive oder »lernende« Ansätze.

Ein individuelles Kostenmodell erfordert die Festlegung von Kostenfunktionen für alle Operationen, die vom Wrapper unterstützt werden. Für jeden Wrapper wird dabei ein eigenes Modell definiert, gegebenenfalls auf Basis eines vorgegebenen Standardmodells [RoÖH99]. Dies ermöglicht eine detaillierte Kontrolle, erhöht jedoch den Aufwand der Wrapper-Entwicklung. Da außerdem häufig Quellen und Wrapper von verschiedenen Personen oder Organisationen entwickelt werden, liegen nur selten exakte Informationen zum Quellsystem vor.

Einen Ausweg bietet daher die Verwendung eines generischen Kostenmodells für alle Wrapper und Quellen, welches durch gezielte Testanfragen »kalibriert« wird. So lassen sich nicht nur Informationen über die Daten (Relationengröße, Werteverteilung etc.) ermitteln, sondern auch über die mit den einzelnen Operationen verbundenen Kosten.

Adaptive Techniken stellen schließlich die konsequente Weiterführung des Kalibrierungsansatzes dar. Hierbei wird versucht, durch die Beobachtung der Anfrageausführung Rückschlüsse auf die Kosten für Operationen und die Charakteristik der Daten zu ziehen. Wird während der Ausführung einer Anfrage festgestellt, dass die Selektivitätsabschätzung zu ungenau war, kann ein Korrekturfaktor berechnet und zur Optimierung späterer Anfragen genutzt werden [SLMK01].

6.4.4 Anfrageausführung

In der letzten Phase der Anfragebearbeitung wird der erstellte Plan abgearbeitet, indem die Teilanfragen an die einzelnen Quellen gesendet und die Ergebnisse eingesammelt werden. Diese sind unter Anwendung weiterer Operationen wie Verbund, Vereinigung, Selektion und Projektion zu kombinieren bzw. weiterzuverarbeiten und schließlich dem Nutzer oder der Anwendung zur Verfügung zu stellen. Als spezielles Problem bei Anfragen an Quellen im Internet können sich die unter Umständen stark variierenden Bearbeitungszeiten – auch für ein und dieselbe Anfrage – erweisen. Diese äußern sich in Verzögerungen bis zum Eintreffen der ersten Ergebnisse oder in unterschiedlichen Ankunftsraten und können zu einer Blockierung der Anfrageausführung führen. Als Beispiel sei die Berechnung des Verbundes zwischen zwei Quellen auf Basis des Nested-Loops-Join angeführt. Verzögert sich beim Durchlauf der äußeren Relation das Eintreffen weiterer Tupel der inneren Relation, so blockiert die gesamte Operation, obwohl eigentlich mit den bereits vorhandenen Tupeln der inneren Relation und den noch nicht verarbeiteten Tupeln der äußeren Relation weitergearbeitet werden könnte.

Eine Lösung dieses Problems ist auf zwei Ebenen möglich:

- auf Operatorebene, d.h. durch spezielle Implementierungen z.B. des Verbundes,
- auf Anfrageebene, indem versucht wird, den Plan so zu modifizieren, dass zunächst andere Operationen ausgeführt werden, bis die Blockierung aufgehoben ist.

Eine spezielle Verbundmethode für solche Szenarien ist der so genannte *XJoin* [UrFr00], der auf dem symmetrischen Hash-Join paralleler Datenbanksysteme aufbaut. Die Idee ist hierbei das Verwalten von jeweils einer eigenen Hashtabelle für die Eingaberelationen. Ein eintreffendes Tupel wird in die eigene Hashtabelle eingefügt und sofort mit den Einträgen in der anderen Hashtabelle auf Erfüllung der Verbundbedingung geprüft. Dadurch kann auch dann weitergearbeitet werden, wenn sich das Eintreffen der Tupel aus einer Quelle verzögert. Der XJoin erweitert dieses Prinzip noch um ein Partitionierungsschema zum Auslagern von Teilen der Hashtabellen auf den externen Speicher. Darüber hinaus wird noch eine asynchrone Komponente eingeführt, die das Einbringen ausgelagerter Partitionen für den Fall steuert, dass beide Quellen blockiert sind.

Auf Anfrageebene kann auf Verzögerungen der Antwort der Quellen durch Änderungen an der Abfolge der Operationen reagiert werden. Eine konkrete Technik hierfür ist das *Query Scrambling* [UrFA98], das zwei Phasen umfasst. Den Ausgangspunkt bildet dabei ein Plan, der in traditioneller Weise optimiert wurde. Kommt es zur Verzögerung der Ausführung einer Operation, so wird die erste Phase, das so genannte *Rescheduling*, eingeleitet. Hierbei wird versucht, zunächst nur die Reihenfolge der Ausführung einzelner Operationen zu modifizieren, ohne dass dadurch die grundsätzliche »Form« des Anfragebaums verändert wird, d.h., die Verbundreihenfolge bleibt erhalten. Dazu werden nicht-blo-

ckierte Teilpläne gesucht und zwischen den Wurzelknoten eines solchen Plans und dessen Elternknoten jeweils ein Materialisierungsoperator eingefügt, der eine Fortsetzung der Ausführung des Teilplanes erlaubt. Kann schließlich auch Phase 1 nicht mehr fortgesetzt werden, so erfolgt der Übergang zur *Operator-Synthesis*-Phase, in der neue, ausführbare Operationen in den Plan eingefügt werden. Ein Beispiel hierfür ist ein Verbund zwischen zwei Relationen, die zuvor nicht direkt verbunden waren. Für einen konkreten Plan $((A \bowtie B) \bowtie C) \bowtie D$ könnte bei einer Verzögerung beim Lesen von B versucht werden, den Verbund $A \bowtie C$ zu berechnen, und erst später bei Verfügbarkeit von B den Verbund $(A \bowtie C) \bowtie B$.

Eine verallgemeinerte Form dieser Technik ist die dynamische Reoptimierung [KaDW98], die ursprünglich für Fälle entwickelt wurde, wo Selektivitätsschätzungen im Rahmen der Optimierung zu fehlerhaften Kardinalitätsannahmen der Relationen führen. Dieses Verfahren basiert auf der Idee, nach jedem Abschluss einer blockierenden Operation (wie z.B. Sortierung, Aufbau einer Hashtabelle etc.) den verbleibenden Teil des Plans neu zu optimieren und dazu die aktuellen und genauen Informationen zur Größe der Zwischenrelationen zu nutzen.

Abschließend sei angemerkt, dass die hier vorgestellten Techniken als sich ergänzende Bausteine der Anfragebearbeitung in Mediatorsystemen zu verstehen sind. Anfrageoptimierung und –ausführung in hochgradig verteilten und heterogenen Systemen stellen ein aktuelles Forschungsgebiet dar, und keine der heute verfügbaren Produkt- und Prototypimplementierungen nutzt diese Techniken in ihrer vollständigen Breite aus.

6.5 Zusammenfassung

Die Integration von Daten aus dem Web eröffnet eine Vielzahl neuer Anwendungen, beginnend bei einfachen Such- oder Shopping-Diensten über personalisierte Portale bis hin zu komplexen Business-Intelligence-Anwendungen, die es ermöglichen, aus der Kombination der Daten und deren Analyse einen Mehrwert abzuleiten. Neben der einfachen Extraktion der Daten aus den Quellen und der Sammlung und Materialisierung in einer Data-Warehouse-ähnlichen Datenbank bietet sich für viele Anwendungsszenarien ein Mediator-basierter Ansatz zur virtuellen Integration an. Die hierfür notwendigen Methoden und Techniken haben wir in diesem Kapitel vorgestellt. So bieten Techniken der Schemaintegration und -transformation Unterstützung bei der Ableitung eines globalen Mediatorschemas und dessen Abbildung auf die Quellschemata. Spezielle Optimierungstechniken und Operatoren berücksichtigen die Besonderheit der Kommunikation im Web sowie der Datenquellen bei der Verarbeitung von Anfragen.

Aktuelle Entwicklungen in diesen Bereichen sind auf eine verbesserte Unterstützung bei Schemaentwurf und –abbildung sowie der Einbeziehung der Quellen gerichtet. Darüber hinaus verspricht die Nutzung von Hintergrundwissen nicht nur eine Vereinfachung der Integration, sondern auch verbesserte Such- und Navigationsmöglichkeiten. Schließlich zielen neue Architekturkonzepte wie etwa ver-

teilte Mediatoren und neue Techniken der Anfragebearbeitung auf eine bessere Skalierbarkeit für Szenarien mit sehr großen Nutzerzahlen ab.

Literatur

[AbBS00] S. Abiteboul, P. Buneman, D. Suciu. *Data on the Web – From Relations to Semistructured Data and XML*. Morgan Kaufmann Publ., 2000.

[AlCo01] E. Altareva, S. Conrad. *The Problem of Uncertainty and Database Integration*. In R.-D. Kutsche, S. Conrad, W. Hasselbring, editors, Engineering Federated Information Systems, Proceedings of the 4th Workshop EFIS 2001, Oct 9-10, 2001, Berlin (Germany), pages 92-99. infix-Verlag / IOS Press, 2001.

[AhMN94] H. Ahonen, H. Mannila, N. Nikunen. *Generating Grammars for SGML Tagged Texts Lacking DTD*. In Proc. PODP'94 – Workshop on Principles of Document Processing, 1994.

[Conr97] S. Conrad. *Föderierte Datenbanksysteme: Konzepte der Datenintegration*. Springer-Verlag, Berlin/Heidelberg, 1997.

[Dada96] P. Dadam. *Verteilte Datenbanken und Client/Server-Systeme. Grundlagen, Konzepte und Realisierungsformen*. Springer Verlag, 1996.

[DuSD95] W. Du, M.-C. Shan, U. Dayal. *Reducing Multidatabase Query Response Time By Tree Balancing*. In Proc. of ACM SIGMOD Conf. On Management of Data, San Jose, CA, 1995.

[EHSS00] M. Endig, M. Höding, G. Saake, K. Sattler, E. Schallehn. *Federation Services for Heterogeneous Digital Libraries Accessing Cooperative and Non-cooperative Sources*. In Proc. of Kyoto Int. Conference on Digital Libraries: Research and Practice, IEEE Computer Society Press, 2000.

[ElVE02] M.G. Elfeky, V.S. Verykios, A.K. Elmagarmid. *TAILOR: A Record Linkage Toolbox*. In Proc. Int. Conf on Data Engineering (ICDE'2002), IEEE CS Press, 2002.

[FeSu69] I.P. Fellegi, A.B. Sunter. *A Theory for Record Linkage*. Journal of the American Statistical Association, Vol. 64, pp. 1183-1210, 1969.

[FlLM98] D. Florescu, A. Levy, A. Mendelzon. *Database Techniques for the World-Wide Web: A Survey*. ACM SIGMOD Record, Vol. 27, No. 3, pp. 59-74, 1998.

[Grae93] G. Graefe. *Query Evaluation Techniques for Large Databases*. ACM Computing Surveys, Vol. 25, No. 2, pp. 73-170, 1993.

[Hale01] A. Halevy. *Answering Queries Using Views*. VLDB Journal, Vol. 10, No. 4, pp. 270-294, 2001.

[HäRa01] T. Härder, E. Rahm. *Datenbanksysteme – Konzepte und Techniken der Implementierung*, 2. Auflage, Springer-Verlag, 2001.

[Hass99] W. Hasselbring. *Top-Down vs. Bottom-Up Engineering of Federated Information Systems*. In S. Conrad, W. Hasselbring, G. Saake, editors, Engineering Federated Information Systems, Proceedings EFIS'99, infix-Verlag, pp. 131-138, 1999.

[HöCo98] M. Höding, S. Conrad. *Data-Mining Tasks in Federated Database Systems Design*. In Issues and Applications of Database Technology (IADT'98), Proc. of the 3rd World Conf. on Integrated Design and Process Technology, 1998, Berlin, Germany, 1998.

[HeSt98] M.A. Hernández, S.J. Stolfo. *Real-World Data is Dirty: Data Cleansing and the Merge/Purge Problem*. Data Mining and Knowledge Discovery, Vol. 2, No. 1, pp. 9-37, 1998.

[HFLP89] L. Haas, J.-C. Freytag, G. Lohman, H. Pirahesh. *Extensible Query Processing in Starburst*. In Proc. of ACM SIGMOD Conf. On Management of Data, Portland, OR, 1989.

[HKWY97] L. Haas, D. Kossmann, E. Wimmers, J. Yang. *Optimizing Queries across Diverse Data Sources*. In Proc. of the Int. Conf. On Very Large Data Bases (VLDB), Athens, Greece, 1997.

[KaDW98] N. Kabra, D. DeWitt. *Efficient Mid-Query Re-optimization of Sub-optimal Query Execution Plans*. In Proc. of ACM SIGMOD Conf. On Management of Data, Seattle, WA, 1998.

[Koss00] D. Kossmann. *The State of the Art in Distributed Query Processing*. ACM Computing Surveys, Vol. 32, No. 4, pp. 422-469, 2000.

[LeRO96] A. Levy, A. Rajaraman, J. Ordille. *Querying Heterogeneous Information Sources Using Source Descriptions*. In Proc. of the Int. Conf. On Very Large Data Bases (VLDB), Bombay, India, 1996.

[LuGM01] B. Ludäscher, A. Gupta, M. Martone. *Model-based Mediation with Domain Maps*. In Proc. of Int. Conf. On Data Engineering (ICDE), Heidelberg, 2001.

[PaGW95] Y. Papakonstantinou, H. Garcia-Molina, J. Widom. *Object Exchange Across Heterogeneous Information Sources*. In Proc. of Int. Conf. On Data Engineering (ICDE), Taipeh, Taiwan, 1995.

[PiBE95] E. Pitoura, O. Bukhres, A. Elmagarmid. *Object-Orientation in Multidatabase Systems*. ACM Computing Surveys, 27(2):141-195, 1995.

[PGGU95] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, J. Ullman. *A Query Translation Scheme for Rapid Implementation of Wrappers*. In Proc. of Conf. on Deductive and Object-Oriented Databases (DOOD), 1995.

[RaBe01] E. Rahm, P.A. Bernstein. *A Survey of Approaches to Automatic Schema Matching*. VLDB Journal, 10(4):334-350, 2001.

[RaDo00] E. Rahm, H.H. Do. *Data Cleaning: Problems and Current Approaches*. IEEE Data Engineering Bulletin 23(4): 3-13, 2000.

[RoÖH99] M.T. Roth, F. Özcan, L.M. Haas. *Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System*. In Proc. of. the Int. Conf. On Very Large Data Bases (VLDB), Edinburgh, Scottland 1999.

[SaHe99] G. Saake, A. Heuer. *Datenbanken – Implementierungstechniken*. MITP-Verlag, Bonn, 1999.

[SLMK01] M. Stillger, G. Lohmann, V. Markl, M. Kandil. *LEO – DB2's LEarning Optimizer*. In Proc. of the Int. Conf. On Very Large Data Bases (VLDB), Roma, Italy 2001.

[SpPD92] S. Spaccapietra, C. Parent, Y. Dupont. *Model Independent Assertions for Integration of Heterogeneous Schemas*. VLDB Journal, 1(1):81-126, 1992.

[UrFA98] T. Urhan, M. Franklin, L. Amsaleg. *Cost Based Query Scrambling for Initial Delays*. In Proc. of ACM SIGMOD Conf. On Management of Data, Seattle, WA, 1998.

[UrFr00] T. Urhan, M. Franklin. *XJoin: A Reactively-Scheduled Pipelined Join Operator*. IEEE Data Engineering Bulletin Vol. 23, No. 2, pp. 27-33, 2000.

[VaPa97] V. Vassalos, Y. Papakonstantinou. *Describing and Using the Query Capabilities of Heterogeneous Sources*. In Proc. of the Int. Conf. On Very Large Data Bases (VLDB), Athens, Greece, 1997.

[Wied92] G. Wiederhold. *Mediators in the Architecture of Future Information Systems*. IEEE Computer, Vol. 25, No. 3, pp. 38-49, 1992.

[YuMe98] C.T. Yu, W. Meng. *Principles of Database Query Processing for Advanced Applications*. Morgan Kaufmann Publ., 1998.

7 Web Caching

Gerhard Weikum

Kurzfassung

Dieses Kapitel gibt einen Überblick über Architektur, Mechanismen und Strategien für Web Caching. Es wird diskutiert, in welchen Rechnern im Netzwerk zwischen Client und Daten-Server Caching möglich und sinnvoll ist, welche Daten in – platzbeschränkten – Caches gehalten werden sollten, unter welchen Bedingungen und wann Prefetching-Maßnahmen zum Vorladen von Daten angemessen sind und wie man in einer solchen verteilten Caching-Architektur die Aktualität von Datenkopien sicherstellt.

7.1 Motivation und Grundbegriffe für Web Caching

Eine frustrierende Erfahrung, die viele Benutzer von Informationsangeboten und elektronischen Diensten des Web gelegentlich machen müssen, ist das lange Warten auf angeforderte Webseiten – ein Phänomen, das auch als »WWW = World Wide Wait« bekannt ist. Die möglichen Ursachen für unakzeptable Antwortzeiten sind vielfältig: ungenügende Bandbreite der Netzanbindung des Clients (z.B. eines PCs mit Modem-Zugang von zu Hause), überlastete Backbone-Leitungen, ungünstiges Routing von IP-Paketen, unterkonfigurierte HTTP-Server (Web-Server) auf der Dienstanbieterseite, überlastete Backend-Daten-Server usw. Empirischen Studien zufolge [BBK00, LGS00] gibt es im E-Commerce und ähnlichen Anwendungen eine psychologische »Schmerzgrenze« für die vom Benutzer tolerierte Antwortzeit: Web-Dienste, deren Seitenaufbau länger als 7 Sekunden dauert, werden mit hoher Wahrscheinlichkeit von potenziellen Kunden als unzuverlässig eingestuft und künftig nicht mehr aufgerufen.

Eine der wichtigsten Maßnahmen zur Verbesserung der Antwortzeiten im Web ist das *Caching* von Daten in Rechnern, die bezüglich der Zugriffszeiten »näher« an den anfordernden Clients liegen. Dabei bedeutet Caching, dass wichtige, d.h. häufig oder in jüngster Vergangenheit benutzte Daten als temporäre Kopien auf diesen nahen Rechnern lokal gespeichert werden. Nähe bezüglich der Zugriffszeiten kann sich auf die Latenzzeiten beziehen, was typischerweise auch Nähe in der Netztopologie impliziert (z.B. ein Rechner im selben LAN), oder auf die Transferraten, wobei auch geographisch weit entfernte Rechner attraktiv sein können. Wenn ein Client auf ein Datenobjekt zugreifen möchte, wird zuerst geprüft, ob

dieses temporär im Speicher oder auf Platten eines nahen Rechners, also in einem *Cache*, liegt und von dort geladen werden kann, anstatt das Objekt von seinem (durch seine URL spezifizierten) Originalspeicherplatz, seinem *Heimatrechner* oder kurz seiner *Heimat (Home)*, anzufordern. Bei einem solchen *Cache-Treffer* wird also die Zugriffszeit auf das Objekt signifikant reduziert. Dies gilt selbst, wenn gegebenenfalls noch beim Heimatrechner nachgefragt werden muss, ob die im Cache gefundene Kopie noch aktuell ist. Ein derartiger Aktualitätstest ist nämlich mit dem HTTP-Auftrag *Get-If-Modified-Since* mit minimalem Datentransfer realisierbar: Wenn sich das Objekt seit dem beim Aufruf als Parameter mitgelieferten Zeitpunkt verändert hat, wird eine aktuelle Kopie vom Server gesendet; andernfalls wird nur eine kurze Bestätigung geschickt, dass die Kopie des Objekts weiterhin gültig ist. Falls der Client bereits früher einmal auf das Objekt zugegriffen hat, wäre offensichtlich der Client selbst (also der eigene PC) der bestmögliche Cache-Rechner. Es gibt jedoch noch wesentlich mehr Möglichkeiten, Caches im Internet zu platzieren.

Beim Caching im engeren Sinne wird erst versucht, ein Objekt in einem Cache zu lokalisieren, wenn es von einem Client explizit angefordert wird. Unter Umständen kann es aber auch sinnvoll sein, schon vorab und ggf. spekulativ ein Objekt zu laden, so dass es, wenn der Client wirklich darauf zugreifen will, bereits ohne jede Zeitverzögerung im (Client-) Cache vorhanden ist. Diese Technik heißt *Prefetching* oder *Vorladen*.

Eine weitere wichtige Technik im Web ist die *Replikation* von Daten auf mehreren Rechnern, häufig auch *Spiegeln (Mirroring)* genannt. Im Gegensatz zum Caching, bei dem Datenobjekte sich u.U. nur kurze Zeit in Caches befinden und dann durch andere, wichtigere Objekte verdrängt werden, wird unter Replikation meist eine längerfristige Datenplatzierungsmaßnahme verstanden. Ein Systemadministrator könnte etwa festlegen, dass bestimmte Webseiten bzw. ganze Verzeichnisse eines Web-Servers für die nächsten Monate auf festgelegten Rechnern (sog. Mirror-Sites) repliziert werden und dass alle Replikate einmal täglich aktualisiert werden. Wenn die Auswahl der replizierten Daten und der Replikationsrechner jedoch selbst durch automatische Algorithmen in Abhängigkeit von zeitlich variablen Zugriffs- und Lastprofilen gesteuert werden, geht eine solche dynamische Form der Replikation gleitend über in ein Caching-Verfahren. Eine derartige Technik wendet zum Beispiel die Firma Akamai für ihr weltweites Content-Delivery-Netz an. Insofern sind also Caching, Prefetching und Replikation nur unterschiedliche Nuancierungen desselben Grundprinzips der *dynamischen Datenreplikation*. Im Folgenden verwenden wir die Begriffe (dynamische) Replikation und Caching als Synonyme.

Die entscheidenden Fragen hinsichtlich der Strategien für dynamische Replikation haben die Form *Wo-Was-Wann-Wie-Warum*. Die *Warum*-Frage wurde bereits in dieser Einleitung beantwortet. Die anderen Fragen beschäftigen sich mit folgenden Aspekten:

- *Wo*, also auf welchen Rechnern, sollen Datenobjekte repliziert werden?
- *Was* soll repliziert werden? Welche Arten von Datenobjekten sind relevant, und nach welchen Kriterien soll die Wichtigkeit von Objekten beurteilt werden, wenn aus Platzgründen Objekte aus einem Cache entfernt – man sagt auch »verdrängt« – werden müssen?
- *Wann* sollen Datenobjekte in einen Cache geladen werden? Offensichtlich ist dies besonders bei Prefetching eine kritische Frage.
- *Wie* sollen Daten auf ihre Aktualität getestet und ggf. aktualisiert werden? Dieser Aspekt ist in der Literatur auch unter dem Stichwort der Kohärenz von verteilten Caches bekannt.

Caching-Techniken und die angesprochenen Wo-Was-Wann-Wie-Warum-Aspekte sind natürlich viel älter als das Web und spielen seit mehreren Jahrzehnten in Betriebssystemen, Datenbanksystemen und verteilten Dateisystemen eine wichtige Rolle. Auf diesen etablierten Techniken baut Web Caching wesentlich auf, adressiert aber eben auch eine Reihe Web-spezifischer Fragestellungen.

Der Rest des Kapitels ist wie folgt aufgebaut. Abschnitt 7.2 diskutiert die Möglichkeiten, auf welchen Rechnern repliziert werden soll, also die Frage der Architekturen für Web Caching. Abschnitt 7.3 stellt Strategien für Cache-Ersetzungen vor und beantwortet damit die Frage, was im Cache gehalten werden soll. Abschnitt 7.4 erweitert dieses Thema auf (spekulatives) Prefetching und geht damit die Frage an, wann ein Datenobjekt in einen Cache vorgeladen werden soll. Abschnitt 7.5 beschäftigt sich mit der Strategien für Cache-Kohärenz, also mit der Frage, wie Daten in verteilten Caches aktuell gehalten werden. Abschnitt 7.6 schließlich beendet dieses Kapitel mit einer Zusammenfassung und einem Ausblick auf weiterführende Themen und aktuell offene Forschungsaspekte.

7.2 Caching-Architekturen: Wo soll repliziert werden?

Für die Platzierung von Caches kommen grundsätzlich alle Rechner auf dem Weg zwischen einem Client und der Heimat eines angeforderten Datenobjekts in Frage. Um die verschiedenen Alternativen besser einordnen zu können, betrachten wir zunächst kurz die Architektur typischer Web-Dienste. Ein *Client* lokalisiert einen Dienst oder ein Datenobjekt zunächst über dessen URL, die mittels einer DNS-Anfrage (Domain Name Service) in eine IP-Adresse umgewandelt wird. Mit dem HTTP-Aufruf Get wird dann das gewünschte Objekt angefordert. Dabei werden häufig alle temporär aufgebauten TCP-Verbindungen über einen *Proxy-Server* der lokalen Organisation (z.B. eines Universitätscampus) oder des Internet-Providers abgewickelt. Der Get-Aufruf kommt initial bei einem *HTTP-Server (Web-Server)* an (der z.B. mit der Apache-Software realisiert ist). Dieser Server kann u.U. dynamisch unter mehreren möglichen Server-Rechnern ausgewählt werden, wenn der mittels IP adressierte Router eine entsprechende lastverteilende Adressauflösung unterstützt. Solche Router werden salopp auch als »IP-Sprayer« bezeichnet und sind bei Web-Anwendungen mit hoher Last üblich.

Wenn das angeforderte Objekt eine statische HTML-Seite ist, wird es in der Regel direkt vom HTTP-Server zurückgeliefert. Handelt es sich um eine dynamische Seite, wird ein Skript oder Servlet aufgerufen, das den Inhalt der Seite berechnet; diese Programme (z.B. PHP- oder Java-Servlets oder sog. Active Server Pages) laufen unter der Kontrolle eines *Applikations-Servers* bzw. einer Servlet-Engine (z.B. mit der PHP- oder der Tomcat-Software). Der Applikations-Server kann bei hoher Last mehrfach auf verschiedenen Rechnern instanziiert sein. Wenn die Konstruktion dynamischer HTML-Seiten Daten aus einer Datenbank benötigt, sprechen die Programme, die auf dem Applikations-Server laufen, selbst einen oder mehrere *Backend-Daten-Server*, typischerweise SQL-Datenbanken, an, und zwar meist über das Protokoll ODBC bzw. JDBC.

In dieser mehrstufigen Architektur auf Seiten des Web-Dienstanbieters kann schließlich auch noch eine weitere Front-End-Komponente vor die initialen HTTP-Server vorgeschaltet werden. Diese wird als *Reverse-Proxy* oder *Web-Server-Accelerator* bezeichnet. Ihre Rolle ist es, wichtige und häufige Sonderfälle mit geringem Overhead direkt zu behandeln und nur die restlichen Fälle an die allgemeinen HTTP-Server weiterzugeben [SILD02, Anto02]. Alle Stufen der Server-Seite können zur Lastverteilung mehrfach instanziiert sein, z.B. innerhalb eines Rechner-Clusters; einen derartigen Komplex nennt man auch eine *Server-Farm*.

Alle Rechner zwischen Client und Backend-Daten-Server sind Kandidaten für Caches. Dies führt folglich auf ein *hierarchisches Caching* entlang des Pfades

Client $\leftrightarrow$ *Proxy* $\leftrightarrow$ *Reverse-Proxy* $\leftrightarrow$ *HTTP-Server*
$\leftrightarrow$ *Applikations-Server* $\leftrightarrow$ *Daten-Server.*

Bei einer Datenanforderung entlang dieses Pfades wird auf jeder Stufe geprüft, ob die Anforderung mit Hilfe des jeweiligen Caches erfüllt werden kann, und der nachfolgende Teil des Pfades wird im Trefferfall ausgelassen. Wenn die Aktualität der im Cache gefundenen Kopie für den Benutzer wichtig ist, wird ggf. noch ein HTTP-Auftrag Get-If-Modified-Since an den HTTP-Server der Heimat generiert.

Wenn Komponenten entlang des hierarchischen Zugriffspfads mehrfach instanziiert sind, kann der Fall eintreten, dass die ausgewählte Instanz einer Stufe, z.B. der HTTP-Server einer Server-Farm, keinen Cache-Treffer hat, das betreffende Datenobjekt aber im Cache einer anderen Instanz derselben Stufe, also gewissermaßen eines »Geschwister-Servers«, vorhanden ist. Um die Kopie dort zu lokalisieren und den Restpfad abschneiden zu können, wird ein Verfahren für *verteiltes, kooperatives Caching* über alle Rechner derselben Stufe benötigt. Da Geschwister-Server in der Regel über ein extrem schnelles Netzwerk (z.B. 1GB-Ethernet oder Myrinet innerhalb eines Rechner-Clusters) verbunden sind, ist es in einer solchen Konfiguration oft sinnvoll, Redundanz zwischen den verschiedenen Caches so weit wie möglich zu vermeiden. Wenn also ein Datenobjekt in einem der Caches vorhanden ist, sollten die Geschwister-Caches möglichst keine weitere Kopie desselben Objekts halten, sondern ihren Speicherplatz eher für das Caching weiterer Objekte investieren. Wenn jedoch das Netzwerk zwischen den Server-

Rechnern (temporär) überlastet ist und zum Engpass wird, ist es gerade wieder sinnvoll, dasselbe Objekt in mehreren Caches zu replizieren. Insofern sollten also solche Geschwister-Caches wie ein größerer Gesamt-Cache mit dynamisch-adaptiven, verteilten Caching-Strategien verwaltet werden. Solche Strategien sind in [DWAP94, LWY96, SiWe97, VLN98, Wolm99, YWM00] näher ausgeführt.

Über diese Fälle hinaus können weitere dedizierte Cache-Server an der »Kante« des Internets sinnvoll sein. Diese sog. *Edge-Server* sollten an strategischen Punkten der Netztopologie vorgesehen werden, eben möglichst nahe bei den Clients mit hoher Auftragsfrequenz. Dabei sollte es mindestens einen solchen Edge-Server im Teilnetz jedes wichtigen Internet-Providers geben, da Übergänge von einem Internet-Provider zu einem anderen häufig leistungsmäßig neuralgische Punkte sind. Die Edge-Server liegen zwar nahe bei den Clients, aber a priori nicht unbedingt auf dem »natürlichen« Routing-Pfad vom Client zum Backend-Daten-Server. Um von deren Caches zu profitieren, ist also eine Umlenkung der Client-Aufträge notwendig. Eine elegante Art, dies zu realisieren, besteht darin, die Edge-Server im DNS zu registrieren und bei der Auflösung von URLs in IP-Adressen jeweils geeignete Edge-Server auszuwählen. Kandidaten sind offensichtlich solche Edge-Server, die die gewünschten Datenobjekte mit hoher Wahrscheinlichkeit repliziert halten. Wenn sich dann wider Erwarten herausstellt, dass der ausgewählte Edge-Server keinen Cache-Treffer hat, muss der Edge-Server die Anforderung geeignet weiterleiten. Diese Technik, Zuteilung mittels DNS mit entsprechender Lastverteilung und Weiterleitung im Nichttrefferfall aufgrund von globalem Wissen über die Cache-Inhalte aller Edge-Server, ist z.B. im Content-Delivery-Network von Akamai (http://www.akamai.com) realisiert [Karg99].

Abbildung 7-1 fasst die möglichen Cache-Platzierungsvarianten in illustrativer Form zusammen. Exzellente Darstellungen dieser Caching-Architekturen und ihrer technischen Details finden sich in den Büchern von Krishnamurthy und Rexford [KrRe01] und Rabinovich und Spatscheck [RaSp02], eine auf aktuelle Produkte eingehende Übersicht ist [Moha01].

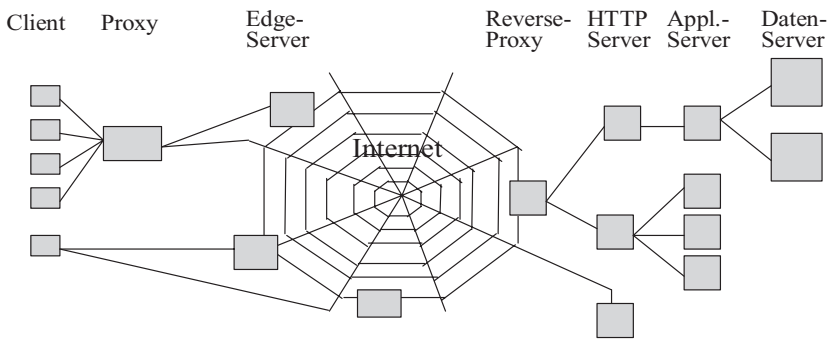


Abb. 7-1: *Hierarchische und verteilte Caching-Varianten im Web*

Bezüglich der in Caches gehaltenen Objekte sind wir bisher implizit davon ausgegangen, dass dies statische HTML-Seiten sind; es können jedoch auch problemlos statische Fragmente dynamischer HTML-Seiten in den verschiedenen Caching-Stufen repliziert werden, z.B. Bilder, Videoclips usw. Gerade Letzteres ist offensichtlich besonders leistungssteigernd und steht z.B. in Content-Delivery-Netzen wie Akamai im Vordergrund.

Darüber hinaus können sogar dynamische HTML-Seiten oder dynamische Fragmente solcher Seiten repliziert werden [CID99, YFIV00]. Ein Cache-Treffer spart in diesem Fall nicht nur den längeren Zugriffsweg, sondern auch die erneute Ausführung des Programms (Skripte, Servlets), mit dem die Seite konstruiert wurde, und ggf. sogar die entsprechenden Datenbankzugriffe. Allerdings fällt dafür der Test, ob eine solche materialisierte Ergebniskopie einer an sich dynamischen Berechnung noch aktuell ist, komplizierter aus. Es ist nämlich zu testen, ob sich Eingabedaten, die den Aufbau der dynamischen Seite beeinflussen könnten, seit der letzten Materialisierung verändert haben. Ist dies der Fall, so müssen Kopien invalidiert werden, und die Programme bzw. Datenbankzugriffe zur Konstruktion der Seite müssen neu ausgeführt werden. Wegen dieses größeren Aufwands für den Aktualitätstest und die eventuell notwendige Aktualisierung werden dynamische Inhalte von Webseiten meist nur in der Nähe der Heimat der entsprechenden Daten in Caches repliziert, also innerhalb des Server-Komplexes des Dienstanbieters, beispielsweise im Reverse-Proxy. Dabei kann nochmals differenziert werden zwischen dem Caching von Datenbankanfrageresultaten [Dar96, SSV96, DILR00, LuNa01, YFIV00] und dem Caching ganzer Programmausführungsresultate. Ersteres kann sogar im Datenbank-Server selbst realisiert werden und wird auch von einigen Produkten unterstützt. Der Test, ob eine SQL-Anfrage das materialisierte Ergebnis einer früheren, nicht notwendigerweise identischen, Anfrage wiederverwenden kann, führt allerdings auf das schwierige (im allgemeinsten Fall sogar unentscheidbare) Problem der Anfrageäquivalenz bzw. Anfrageresultatsüberdeckung (Query Containment, Subsumption) [ChHa80, LaYa85]. Insofern können Datenbanksysteme nur eingeschränkte Fälle von Anfrageresultats-Caching gut unterstützen, beispielsweise Anfragen mit aussagenlogischen Filterbedingungen (die aus den Eingaben in ein HTML-Formular generiert werden). Diese können u.U. bereits einen hohen Leistungsgewinn erzielen [LuNa01]. In einigen Datenbankprodukten wird eine simplifizierte Form dieses Caching-Typs für ganze Datenbanktabellen auf der Basis asynchroner Datenreplikationstechniken realisiert (siehe z.B. [Luo02]).

Tabelle 7-1 fasst die Möglichkeiten der Replikation verschiedener Arten von Datenobjekten zusammen; die in der Praxis typischen Varianten sind mit einem Pluszeichen gekennzeichnet.

Caching-Stufe	Cache-Objekte			
	Statische HTML-Seiten	*Bilder, Videos, etc.*	*Dynamische HTML-Seiten*	*Datenbankanfrage-resultate*
Client	+	+		
Proxy	+	+		
Edge-Server		+		
Reverse-Proxy	+	+	+	
HTTP-Server			+	
Appl.-Server			+	+
Daten-Server			+	+

Tab. 7-1: *Caching-Objekte auf verschiedenen Caching-Stufen*

7.3 Cache-Ersetzungsstrategien: Was soll im Cache gehalten werden?

7.3.1 Basisstrategien

Jeder Cache ist in seinem Platz limitiert, so dass bei Anforderung eines neuen Datenobjekts andere Objekte verdrängt, also aus dem Cache entfernt werden müssen. Die bekanntesten Cache-Ersetzungsstrategien, die auch in Datenbanksystemen, Dateisystemen und für Virtual-Memory-Paging verwendet werden [CoDe73, HäRa01], sind *LRU (least recently used)*, *LFU (least frequently used)* sowie Verallgemeinerungen wie *LRFU* [Lee99] und *LRU-k* [OOW93, OOW99]. Wir diskutieren zunächst kurz diese Basisstrategien, als würden sie auf Seiten bzw. Blöcke fester Größe angewandt, und gehen im nachfolgenden Unterabschnitt auf die notwendigen Erweiterungen für Web Caching ein.

Alle Ersetzungsstrategien weisen jedem Objekt, das im Cache gehalten wird, sowie u.U. weiteren Objekten eine *Wichtigkeit (Popularität, Priorität)* zu, die dynamisch aufgrund des Zugriffsverhaltens der jüngeren Vergangenheit geschätzt wird. Diese Schätzung dient zur Extrapolation des Zugriffsverhaltens in der näheren Zukunft, also z.B. zur Schätzung der Wahrscheinlichkeit, dass auf das Objekt in den nächsten 5 Minuten nochmals zugegriffen wird. In die Wichtigkeit können das *Alter* des Objekts, also die Zeit seit dem letzten Zugriff, und die *Häufigkeit* der Zugriffe einfließen; Ersteres führt auf die LRU-Familie, Letzteres auf die LFU-Familie. Kombinierte Verfahren wie LRU-k und LRFU berücksichtigen beide Kriterien bei der Berechnung der Wichtigkeit. Wenn zur Einlagerung eines Objekts Platz in einem ansonsten vollen Cache benötigt wird, dann wird jeweils das aktuell unwichtigste Objekt (das Objekt mit der niedrigsten Priorität) als Ersetzungsopfer ausgewählt.

Die LRU-Strategie wählt unter allen im Cache aktuell gehaltenen Objekten dasjenige als Ersetzungsopfer aus, dessen jüngster Zugriffszeitpunkt am weitesten in der Vergangenheit liegt, also das in diesem Sinne älteste Objekt. Man beachte,

dass sich Alter hier auf den jeweils jüngsten Zugriffszeitpunkt bezieht und nicht auf den Zeitpunkt, zu dem ein Objekt in den Cache geladen wurde; Letzteres würde auf eine FIFO-Strategie führen, die in den meisten Anwendungsfällen für Caching unbrauchbar ist. LRU lässt sich sehr einfach implementieren: Man organisiert einfach alle im Cache gehaltenen Objekte (bzw. eigentlich deren Buchführungsinformationen) in einer doppelt verketteten Liste, die nach den LRU-Zeitpunkten der Objekte sortiert gehalten wird. Beim Zugriff auf ein Objekt wird es von seiner aktuellen Position in der Liste an das hoch priorisierte Ende bewegt, und das jeweilige Ersetzungsopfer findet sich am niedrig priorisierten Ende der Liste. Der Zeit-Overhead für die LRU-Buchführung ist also konstant, unabhängig davon, wie groß der Cache ist.

Die LFU-Strategie führt für jedes Objekt einen Zähler, der die Anzahl der Zugriffe auf das Objekt wiedergibt. Als Ersetzungsopfer wird jeweils das Objekt ausgewählt, dessen Zähler den niedrigsten Wert hat. Der Zeit-Overhead für diese Ersetzungsentscheidungen ist im schlechtesten Fall logarithmisch in der Cache-Größe, da man die Objekte nun in einer Prioritäts-Queue (z.B. einem Fibonacci-Heap oder einem einfachen 2-3-Baum) verwalten muss. LFU wäre ein optimales Cache-Ersetzungsverfahren, wenn die Zugriffswahrscheinlichkeiten auf die verschiedenen Datenobjekte zeitlich invariant (und voneinander unabhängig) wären [CoDe73]. Die LRU-Strategie wäre demgegenüber suboptimal, da sie Objekten, auf die nur einmal, aber gerade eben zugegriffen wurde, fälschlich eine hohe Wichtigkeit zuschreibt. In realen Anwendungen aber sind Zugriffswahrscheinlichkeiten nicht invariant, sondern verändern sich (wenn auch langsam) mit der Zeit. Daraus resultiert der entscheidende Nachteil des LFU-Verfahrens, dass nämlich Objekte, die einmal eine hohe Zugriffshäufigkeit hatten, extrem lange im Cache bleiben, auch wenn sie mittlerweile längst nicht mehr populär sind. Eine reine LFU-Strategie passt sich also nicht oder nur sehr langsam den zeitlichen Veränderungen der Zugriffscharakteristika an. Aus diesem Grund haben LFU-basierte Cache-Implementierungen typischerweise eine Alterung der Häufigkeitszähler eingeführt. Eine Möglichkeit ist, nach jeweils x Objektzugriffen alle Häufigkeitszähler um den Wert y zu reduzieren (siehe z.B. [HäRa01]). Dieses an sich recht leistungsfähige Verfahren hat allerdings den Nachteil, dass die beiden Tuning-Parameter x und y sorgfältig gewählt werden müssen; gerade für hochgradig variable Web-Lasten dürfte dies selbst für hervorragende Systemarchitekten oder Systemadministratoren alles andere als eine triviale Aufgabe sein. Die beiden Ersetzungsstrategien LRFU und LRU-k vermeiden weitgehend solche kritischen Tuning-Parameter bzw. verwenden nur jeweils einen einfacher und robust wählbaren Parameter.

Die LRFU-Strategie [Lee99] ordnet jedem Objekt einen Wichtigkeitswert W zu, der bei der Einlagerung des Objekts zunächst mit 0 initialisiert und bei jedem weiteren Zugriff nach der Formel $W := 1 + \lambda W$ adjustiert wird. Dabei ist λ ein Parameter, der zwischen 0.5 und 1 liegen muss; für $\lambda=1$ erhält man LFU als Spezialfall, und für $\lambda \leq 0.5$ verhält sich das Verfahren wie LRU. Die Implementierung des Verfahrens benötigt offensichtlich wie auch LFU eine Prioritäts-Queue.

Die LRU-k-Strategie [OOW93, OOW99] schließlich integriert LRU- und LFU-ähnliche Überlegungen, indem sie sich für jedes im Cache befindliche Objekt und eine beschränkte Menge zusätzlicher interessanter Objekte (z.B. alle Objekte, auf die es in den letzten δ Zeiteinheiten mindestens einen Zugriff gab) die Zeitpunkte der jeweils k letzten Zugriffe merkt. Seien t0 die aktuelle Zeit und $t_1(x)$, ..., $t_k(x)$ die Zeitpunkte der k jüngsten Zugriffe auf Objekt x. Dann wird die Wichtigkeit von x – hier auch *Hitze (Heat)* genannt [CABK88] – durch folgenden Ausdruck geschätzt: $Heat(x) = (t_0 - t_k(x)) / k$. Mathematisch gesehen ist dies der Maximum-Likelihood-Schätzer für die Zugriffsrate von Objekt x, also proportional zur Wahrscheinlichkeit, dass auf x in einem kleinen Zeitintervall einmal zugegriffen wird. Da sich die aktuelle Zeit t_0 fortlaufend verändert und die ständige Neuberechnung von Heat(x) offensichtlich unakzeptabel wäre, verwendet LRU-k für die Ersetzungsopferauswahl den Wichtigkeitswert $t_k(x) / k$. Dieser gibt die relative Wichtigkeit zwischen verschiedenen Objekten getreu wieder, und bei der Cache-Ersetzungsstrategie kommt es nur auf die relative Wichtigkeit an. Zur Implementierung von LRU-k benötigt man wiederum eine Prioritäts-Queue; das Verfahren ist also vom Overhead etwas aufwändiger als LRU, im Leistungsverhalten aber signifikant überlegen.

Man kann beweisen, dass LRU-k unter allen Caching-Strategien, deren Entscheidungen auf der Kenntnis der letzten k Zugriffszeitpunkte aller Objekte beruhen, optimal ist [OOW99]. LRU ist als Sonderfall in der LRU-k-Familie enthalten, nämlich mit der Wahl k=1. Ein hoher Wert für k (z.B. k=10) führt zu einer hohen Konfidenz der Schätzung für die Zugriffsrate der Datenobjekte, hat aber offensichtlich auch eine entsprechend höhere Trägheit gegenüber Veränderungen der Zugriffsmuster. Ein kleiner Wert für k (z.B. k=2 oder k=3) erlaubt dem Verfahren dagegen eine schnelle Anpassung an sich verändernde Lastcharakteristika. Für den Einsatz als Cache-Ersetzungsverfahren hat sich die Wahl k=2 bewährt. Wir kommen auf LRU-k aber auch in Verbindung mit spekulativem Prefetching nochmals zurück und würden dafür k eher konservativer, also höher, wählen.

7.3.2 Web-spezifische Strategien

Im Kontext von Web Caching stellen sich zwei wesentliche Zusatzprobleme, mit denen ein leistungsfähiges Caching-Verfahren umgehen können sollte:

- die variable Größe der Datenobjekte (HTML-Dokumente, Bilder, etc.), die bei einer rein seiten- bzw. blockorientierten Strategie unberücksichtigt bliebe, und
- die variablen Zugriffskosten (Latenzzeit, »verbrauchte« Netzwerkbandbreite, Server-CPU-Zeit, Plattenzugriffszeit etc.), die sich ergeben, wenn ein Objekt nicht im Cache ist und von seinem Heimat-Server (oder einem anderen, weiter entfernten Cache in der Caching-Hierarchie) geladen werden muss.

Kommt es bei einer Ersetzungsentscheidung zur Auswahl zwischen zwei Objekten, die bzgl. Alter, Zugriffshäufigkeit, Hitze und den anderen im vorigen

Abschnitt diskutierten Kriterien gleich wichtig sind, dann sollte das kleinere bevorzugt im Cache gehalten werden, also das größere als Opfer gewählt werden. Wenn die beiden Objekte auch gleich groß sind, sich jedoch eines davon auf einem nahe gelegenen Server mit schneller Netzwerkanbindung befindet und das andere auf einem langsamen Server mit schlechter Netzanbindung, sollte das Erstere ersetzt werden, da es sich mit niedrigeren Kosten von seinem Server wiederbeschaffen lässt. Die Entscheidungssituation wird aber offensichtlich sehr viel komplexer, wenn sich die Objekte in allen drei Kriterien – Hitze, Größe und Kosten – unterscheiden und keines das andere in allen Dimensionen dominiert.

Die im vorhergehenden Abschnitt skizzierten Basisstrategien LRU, LRU-k und LRFU sind auch im Kontext von Web-Caches einsetzbar, lassen aber zunächst die variable Größe von Datenobjekten außer Acht. LRU wird sogar in der weit verbreiteten Proxy-Software Squid (http://www.squid-cache.org) verwendet. Eine einfache Ad-hoc-Heuristik zur Berücksichtigung der variablen Größe von Datenobjekten ist die Größenbeschränkung der Objekte, die überhaupt im Cache gehalten werden. Objekte, deren Größe einen vorgegebenen Schwellwert (z.B. 100 KBytes) überschreitet, werden nicht im Cache repliziert, sondern so schnell wie möglich aus dem Speicher der jeweiligen Caching-Stufe entfernt. Mit ähnlichen Ad-hoc-Techniken ließe sich auch unter den kleineren, zum Caching tauglichen Objekten, eine größenabhängige Differenzierung herbeiführen. Ein systematischer Ansatz dagegen beruht darauf, die Wichtigkeit eines Datenobjekts auf die einzelnen Bytes des Objekts umzurechnen, also die Objektwichtigkeit entsprechend der Objektgröße zu normalisieren.

Diese Überlegung lässt sich in eleganter Weise fortführen, wenn man die Hitze eines Objekts x (im Sinne von LRU-k) als Maß für Wichtigkeit zugrunde legt, und führt auf das Maß der *Temperatur* [CABK88]: Temp(x) = Heat(x) / Size(x). Jetzt können wir die Wichtigkeit eines Objekts durch seine Temperatur definieren und als Ersetzungsopfer jeweils das Objekt mit der niedrigsten Temperatur, also Hitze pro im Cache belegtem Byte, auswählen. Man beachte, dass bei einem solchen *temperaturorientierten Ersetzungsverfahren* natürlich nicht etwa einzelne Bytes von Objekten ersetzt werden, sondern immer ganze Objekte; die Normierung der Hitze pro Byte dient nur der Priorisierung zwischen Objekten. Beim Einlagern eines neuen Objekts in den Cache müssen daher u.U. mehrere Objekte verdrängt werden, um ausreichend freien Platz im Cache zu schaffen; diese Opfer werden in aufsteigender Temperaturrangfolge bestimmt. Außerdem ergibt sich u.U. das Problem, dass der Speicherplatz des Caches fragmentiert ist: Es ist insgesamt noch ausreichend freier Platz vorhanden, aber dieser ist nicht zusammenhängend, sondern besteht aus mehreren Fragmenten, von denen jedes einzelne zu klein ist. Dies ist ein Standardproblem der Speicherverwaltung, für das es eine Vielzahl leistungsfähiger heuristischer Lösungen gibt. Das Problem ist aber nur bei sehr kleinen Caches (relativ zur maximalen Objektgröße) gravierend; bei hinreichend großen Caches ist Fragmentierung weitgehend vernachlässigbar.

Temperaturorientiertes Caching löst also das erste der eingangs dieses Abschnitts diskutierten Probleme. Um auch das zweite Problem, das der variablen

Zugriffskosten, zu lösen, besinnen wir uns darauf, dass Caching generell als mathematisches Optimierungsproblem formuliert werden kann: Die bestmögliche Belegung eines Caches – bei zeitinvarianten Last- und Systemparametern – ist diejenige Auswahl an Datenobjekten, für die die mittleren Zugriffskosten auf alle Objekte minimal werden. Dabei fallen für die im Cache gehaltenen Objekte praktisch keine (oder vernachlässigbar kleine) Kosten an und für alle anderen Objekte die Zugriffskosten entsprechend der jeweiligen Heimat-Server (und deren Netzwerkanbindung usw.). Bei der Mittelung über alle Objekte spielen außerdem die objektspezifischen Zugriffshäufigkeiten (geschätzt durch die jeweilige Hitze) die Rolle von Gewichten. Wenn wir schließlich die Hitze pro Byte betrachten, sehen wir, dass wir die Summe

$\Sigma_{\text{alle Objekte x}}$ Temp(x)*Cost(x) minimieren bzw.
$\Sigma_{\text{alle Objekte y im Cache}}$ Temp(y)*Cost(y) maximieren müssen.

Dabei bezeichnen Cost(x) bzw. Cost(y) die jeweiligen objektspezifischen Zugriffskosten. In Szenarien, bei denen sogar dynamische HTML-Seiten oder Datenbankanfrageergebnisse in Caches repliziert werden, beinhalten diese Kosten auch den Aufwand für die erneute Programmausführung zur Konstruktion der dynamischen Seite oder die erneute Ausführung einer Datenbankanfrage.

Für ein praktikables Caching-Verfahren müssen wir allerdings das Optimierungsproblem online lösen, und zwar bei jeder anstehenden Cache-Ersetzungsentscheidung. Sowohl Temperatur als auch Kosten müssen also dynamisch geschätzt werden. Ersteres können wir mit dem LRU-k-basierten Hitzeschätzer lösen, Letzteres durch Online-Statistiken über Netzzugriffslatenzzeiten und -auslastungen, Auslastungen und Antwortzeiten von Server-Systemen usw. Für die Online-Auswahl von Cache-Ersetzungsopfern schließlich definieren wir die Wichtigkeit eines Objekts x als Produkt Temp(x)*Cost(x) = Heat(x)*Cost(x) / Size(x). Dieses Produkt bezeichnen wir als *Nutzen (Benefit)* eines Objekts x und das entsprechende Cache-Ersetzungsverfahren als *nutzenorientiertes Caching*. Im stationären Zustand, also bei zeitinvariantem und lediglich stochastisch fluktuierendem Verhalten, ist dieses Verfahren optimal. Im realen Web, mit zeitlich veränderlichen Last- und Systemcharakteristika, ist nutzenorientiertes Caching die beste bekannte Online-Heuristik. Caching-Algorithmen, die auf diesen Prinzipien beruhen, wurden in verschiedenen Anwendungskontexten (Web, Data Warehouses, Speicherhierarchien) in [LRV96, CaIr97, SSV97, SSV99, CoKa99, JiBe00], [SSV96] und [KrWe97, KrWe98] entwickelt. In der Web-spezifischen Arbeit [CaIr97] wurde diese Strategie als *GreedyDual-Size (GDS)* bezeichnet (als Verallgemeinerung des GreedyDual-Paging-Verfahrens von [Young94]).

7.3.3 Konfiguration der Cache-Größe

Für das Leistungsverhalten eines Caches ist neben der Ersetzungsstrategie vor allem auch eine angemessene Cache-Größe ausschlaggebend. Diesbezüglich gibt es zwei nützliche Faustregeln [GrSh00], die hier kurz skizziert werden:

- die sog. *5-Minuten-Regel*, die sich am Kosten-Leistungs-Verhältnis mit dem Durchsatz der Datenobjektzugriffe als Leistungsmetrik orientiert, und
- eine zweite Regel, die sich am Kosten-Leistungs-Verhältnis mit der Antwortzeit als Leistungsmetrik orientiert.

Zur Erklärung der 5-Minuten-Regel nehmen wir vereinfachend an, dass wir ein Datenobjekt entweder ständig in einem Cache halten oder nie. Im ersten Fall zahlen wir die Kosten C_{cache} für den notwendigen Speicherplatz der entsprechenden Caching-Stufe, im zweiten Fall zahlen wir die Kosten C_{home}, um mit der für das Objekt spezifischen Zugriffsrate λ immer wieder auf die Heimat des Objekts zuzugreifen. Betrachten wir etwa einen Hauptspeicher-Cache innerhalb eines einfachen HTTP-Servers bzw. einen Reverse-Proxy-Cache vor dem HTTP-Server. Bei heutigen RAM-Preisen und einer Objektgröße σ kostet der Cache ungefähr $C_{cache} = \sigma * 1$ Euro/MByte. Auf der Server-Seite, also ohne den Cache, kostet eine leistungsfähige Magnetplatte (inkl. I/O-Bus mit entsprechend hoher Bandbreite) ca. 1000 Euro und kann ca. 100 Zugriffe pro Sekunde ausführen; ein Zugriff pro Sekunde kostet also 10 Euro. Wenn wir – weil der Cache zu klein ist, um das betrachtete Datenobjekt zu replizieren – mit der Rate λ auf die Platte zugreifen müssen, ergeben sich Kosten $C_{home} = \lambda * 10$ Euro/s^{-1}. Es ist vom Kosten-Leistungs-Verhältnis genau dann günstiger, den notwendigen RAM für das Caching des Objekts zu investieren, wenn $C_{cache} < C_{home}$. Durch Einsetzen der o.a. Werte und Auflösen nach der Zugriffsrate λ erhalten wir daraus, dass sich Caching ab $\lambda > 0.1 * \sigma$ $MByte^{-1} * s^{-1}$ lohnt. Wenn das Objekt also z.B. 32 KBytes groß ist, rechtfertigt eine Rate von ungefähr $\lambda > 0.003$ s^{-1} den Cache-Ausbau. Dies bedeutet, dass auf das Objekt im Schnitt einmal alle $1/\lambda \approx 5$ min oder häufiger zugegriffen werden muss; daher der Name der 5-Minuten-Regel. Offensichtlich ist diese Regel aber abhängig von aktuellen Technologieparametern und –preisen. Entsprechend kann sie modifiziert auch auf die Konfiguration eines plattenresidenten Proxy-Caches angewandt werden: C_{cache} ergibt sich dabei aus den Plattenplatzpreisen für den Proxy-Cache, und C_{home} würde zusätzlich zu den I/O-Durchsatzkosten des Servers die Netzwerkkosten von z.B. 1 Euro / GByte Datentransfer für den entfernten Server-Zugriff widerspiegeln.

Die antwortzeitorientierte Faustregel bemisst die Kosten für den Heimatzugriff als Funktion der Zeitverzögerung, die der Benutzer durch den entfernten Zugriff erfährt. Nehmen wir an, diese Latenzzeit beträgt T Zeiteinheiten, beispielsweise 5 Sekunden, und die Kosten für das Warten sind die menschlichen Lohnkosten, z.B. 10 Euro/Stunde. Beziehen wir uns ferner auf 1 Jahr als Amortisationszeitraum für die mögliche Investition in Cache-Platz. Dann betragen die Kosten für den entfernten Zugriff C_{home} = 10 Euro/Stunde * 5 s * λ * 1 Jahr (mit

der objektspezifischen Zugriffsrate λ). Diesen Kosten müssen wir die Cache-Kosten C_{cache} gegenüberstellen, die analog zur 5-Minuten-Regel ermittelt werden, allerdings jetzt mit Plattenplatzkosten $C_{cache} = \sigma$ * 1000 Euro / 100 GByte. Wenn wir dann in die Ungleichung $C_{cache} < C_{home}$ einsetzen (mit einer Objektgröße von σ = 32 KByte) und nach λ auflösen, erhalten wir grob gerechnet $\lambda > 10^{-9}\ s^{-1}$, benötigen also mindestens einen Zugriff in einer Milliarde Sekunden, sprich alle 25 Jahre, um die Investition in das Caching zu rechtfertigen. Das bedeutet praktisch, dass man – wenn überhaupt die Chance besteht, auf ein Datenobjekt in der Zukunft nochmals zuzugreifen – das Objekt im Cache replizieren sollte. Am Cache-Platz zu sparen wäre grundfalsch.

Die antwortzeitorientierte Faustregel suggeriert, dass Caches so groß konfiguriert werden sollten, dass es auf die Ersetzungsstrategien kaum noch ankommt. In der Praxis gibt es allerdings so viele Unwägbarkeiten, insbesondere schnelle Veränderungen in den Zugriffs- und Lastcharakteristika, dass die Regel längst nicht in dem Maße zur Anwendung kommt, wie man dies womöglich aufgrund unserer Reißbrettrechnung erwarten würde. Reale Caches sind in vielen Fällen deutlich kleiner, als die Regel nahelegt. In diesem Fall wäre eine mögliche Konfigurationsrichtlinie, den Cache mindestens so groß zu machen, dass eine bestimmte Wunschtrefferrate, oder besser noch eine vorgegebene obere Schranke für ein Quantil der Antwortzeitverteilung, erreicht wird. Leider ist dies mit einfachen Überschlagsrechnungen wie den oben demonstrierten nicht mehr lösbar. Vielmehr erfordert eine solche angemessene Konfigurierung anspruchsvolle mathematische Modelle zur Vorhersage der Trefferrate und des gesamten Antwortzeitverhaltens bei gegebenen Werten für Cache-Größen, Last-, Netzwerk- und sonstige Systemparameter.

7.4 Prefetching-Strategien: Wann sollen Daten vorgeladen werden?

Beim Caching im engeren Sinne, so wie es im vorhergehenden Abschnitt diskutiert wurde, werden Datenobjekte nur dann in einen Cache eingelagert, wenn sie vom Client angefordert werden; man spricht auch von On-Demand-Caching. Caching hilft also beim allerersten Zugriff auf ein Objekt oder beim ersten Zugriff nach einer längeren Phase, in der das Objekt aus dem Cache entfernt wurde, nicht. Wenn Latenzzeiten (oder die Transferzeiten sehr großer Objekte) für den Benutzer akzeptable »Schmerzgrenzen« überschreiten, scheint ein Schaden (aus Sicht des wartenden Benutzers, aber auch aus Sicht des weniger attraktiv erscheinenden Informations- oder E-Service-Anbieters) unvermeidlich. Die Idee, auch in solchen Fällen die Latenzzeiten zu reduzieren, besteht darin, Objekte schon *vor* der expliziten Anforderung des Benutzers in den Cache zu laden; diese Technik nennt man *Prefetching* oder *Vorladen*. Da man in der Regel nicht im Voraus wissen kann, welche Objekte als nächstes angefordert werden, muss man Prefetching typischerweise *spekulativ* einsetzen. Die Caching-Software muss also

schätzen, auf welche Objekte mit hoher Wahrscheinlichkeit in den nächsten Schritten bzw. in der nahen Zukunft zugegriffen wird, und kann diese dann vorladen. Wenn die Schätzung falsch ist oder der Benutzer unerwartete Folgeschritte auslöst, entsteht aber ein potenzieller Schaden, da zum einen die Vorladetransfers Netzwerkbandbreite konsumieren und damit andere Transfers behindern könnten, und zum anderen die vorgeladenen Objekte Platz im Cache benötigen und damit andere Objekte verdrängen, die sich im Nachhinein als wichtiger erweisen könnten [CFKL95, Patt95]. Mit diesem Spannungsfeld müssen intelligente Prefetching-Strategien geeignet umgehen.

7.4.1 Einfache temperatur- und nutzenorientierte Prefetching-Verfahren

Die einfachste – und eben im obigen Sinne wenig intelligente – Prefetching-Strategie besteht darin, beim Zugriff auf eine HTML-Seite alle darin enthaltenen Hyperlinks zu extrahieren und alle durch diese Links referenzierten Objekte so schnell wie möglich vorzuladen. Dieses *naive Prefetching*, das in den frühen Tagen des Web erwogen wurde, hat sich *nicht* bewährt. Bei Seiten mit sehr vielen ausgehenden Links verschwendet das Vorladen aller Nachfolgerseiten zu viel Bandbreite und serverseitige Ressourcen und behindert damit andere, explizite, Web-Zugriffe so stark, dass der mögliche Nutzen des Vorladens durch den Schaden für andere Zugriffe dominiert wird. Außerdem berücksichtigt dieses Verfahren nur die jeweils unmittelbaren Nachfolgerseiten; bei Seiten, die der Benutzer nur wenige Sekunden anschaut, um dann sofort auf einen weiterführenden Link zu klicken, können Verzögerungen beim Laden der transitiven Nachfolger nicht mehr einfach vermieden werden. Umgekehrt ist aber ein Ausdehnen des naiven Vorladens auf alle transitiven Nachfolger einer bestimmten Tiefe (z.B. alle direkten Nachfolger und deren Nachfolger) praktisch unmöglich, da dies viel zu viele Seiten vorladen müsste und der Schaden durch die Verschwendung der Netzwerk- und Server-Ressourcen explodieren würde.

Die nun naheliegende Idee ist, analog zu den Überlegungen des vorhergehenden Abschnitts, (direkte oder transitive) Nachfolgerseiten nur als Prefetching-Kandidaten zu betrachten und vor dem Vorladen selbst deren Wichtigkeit aufgrund von Statistiken zu schätzen, so dass nur wenige, besonders aussichtsreiche Kandidaten wirklich vorgeladen werden. Wichtigkeit kann dabei – wie beim Caching selbst – auf verschiedene Arten geschätzt werden. Sinnvoll ist z.B. *temperaturorientiertes Prefetching*, bei dem die Wichtigkeit eines Vorladekandidaten durch dessen Temperatur geschätzt wird. Weil Prefetching aber spekulativ und wegen des potenziellen Schadens gewissermaßen riskanter ist als Caching im engeren Sinne, wird man bei der Temperatur- bzw. Hitzeschätzung mit den LRU-k-Statistiken einen höheren und dadurch statistisch signifikanteren Wert von k (z.B. k=10) wählen. Die notwendigen Zugriffsstatistiken wird man wegen des dafür notwendigen Speicherplatzes außerdem nur für eine beschränkte Anzahl von Kandidaten (die eben nicht im Cache sind) führen wollen. Um dies zu implementieren, kann man beispielsweise Statistikeinträge für solche Objekte dyna-

misch entfernen, auf die über einen sehr langen Zeitraum überhaupt nicht mehr zugegriffen wurde. Beim temperaturorientierten Prefetching werden solche Kandidatenobjekte vorgeladen, deren Temperatur einen bestimmten, vom Benutzer bzw. Systemadministrator wählbaren Schwellwert überschreitet, oder – besser, weil ohne einen solchen Tuning-Parameter auskommend – diejenigen Objekte, deren Temperatur die Gesamttemperatur der für das Prefetching aus dem Cache zu entfernenden Opfer überschreitet. Wenn x ein Prefetching-Kandidat ist mit Größe Size(x) und V eine minimale Menge von Opfern mit Gesamtgröße $\Sigma_{v\in V}$-Size(v) ≥ Size(x), dann muss

$$\text{Temp}(x) > (\Sigma_{v\in V}\text{Heat}(v)) / (\Sigma_{v\in V}\text{Size}(v))$$

gelten, damit x vorgeladen wird, und außerdem soll es keine Alternativmenge V' geben mit kleinerer Gesamttemperatur (der rechten Seite der o.a. Ungleichung). Die Implementierung des Verfahrens ist offensichtlich nicht gerade trivial, kann aber mittels geeigneter Prioritäts-Queues und Greedy-Heuristiken effizient realisiert werden [KrWe98].

Der Schritt vom temperaturorientierten Prefetching zum *nutzenorientierten Prefetching* ist – entsprechend den Ausführungen im vorhergehenden Abschnitt – natürlich und geradlinig. Prefetching-Kandidaten werden hinsichtlich ihres Nutzens bewertet und mit den entsprechenden Opfern verglichen. Objekt x wird also nur dann vorgeladen, wenn bzgl. Opfermenge V gilt:

$$\text{Temp}(x) * \text{Cost}(x) > (\Sigma_{v\in V}\text{Heat}(v)*\text{Cost}(v)) / (\Sigma_{v\in V}\text{Size}(v))$$

Die linke Seite der Ungleichung ist der Nutzen, den wir vom Prefetching von x hätten; die rechte Seite gibt den Nutzen an, den wir haben, wenn wir alle Objekte in V weiterhin im Cache behalten (und x eben gar nicht vorladen). Diese Ungleichung berücksichtigt also auch, dass Objekte unterschiedliche Ladezeiten aufweisen, je nachdem, über welche Netzwerkverbindung auf sie zugegriffen wird und wie schnell oder stark belastet die jeweiligen Heimat-Server sind.

Ein Aspekt, der bislang noch nicht explizit berücksichtigt wurde, ist der potenzielle Schaden, den ein Prefetching-Auftrag dadurch auslöst, dass er Netzwerk- und Server-Ressourcen zusätzlich belastet und dadurch die Zugriffszeiten auf explizit angeforderte Objekte verlangsamt [JiKl98, CrBa98, FCLJ99]. Dies ergibt sich aufgrund von Warteschlangenbildung in Routern, Proxies, und Server-Rechnern; eine quantitative Charakterisierung erfordert komplexere stochastische Methoden aus dem Bereich der Warteschlangentheorie. Nehmen wir an, wir können diesen indirekten Schaden, der durch Vorladen von Objekt x entstünde, quantifizieren, und nennen ihn Penalty(x). Damit das Prefetching von x dann noch lohnend ist, muss gelten:

$$\text{Temp}(x) * \text{Cost}(x) - \text{Penalty}(x) > (\Sigma_{v\in V}\text{Heat}(v)*\text{Cost}(v)) / (\Sigma_{v\in V}\text{Size}(v)).$$

Ein Prefetching-Verfahren, das auf diesem quantitativen Kosten-Nutzen-Modell beruht, ist beispielsweise in [KrWe98] beschrieben. In realen Web-Komponenten kommen solche anspruchsvollen Methoden noch nicht zum Einsatz, doch das Leistungspotenzial solcher intelligenter Strategien, das sich z.B. in Simulationsexperimenten zeigt, ist vielversprechend.

Bei all diesen Prefetching-Verfahren kann in die Schätzung der Zugriffsraten, über den Rückgriff auf LRU-k-artige Statistiken hinausgehend, auch zusätzliches Wissen über Benutzerprofile und Webseiteninhalte einfließen. Beispielsweise kann die geschätzte Zugriffsrate auf eine Seite zum Thema Sport in der Buchführung der Prefetching-Software erhöht werden, wenn ein wichtiges Sportereignis stattfindet (z.B. die Olympischen Spiele) oder die Hitzestatistiken für thematisch verwandte Seiten hochgehen. Dies ist ein weites Feld für Heuristiken und experimentelle Forschung mit einem großen Potenzial für benutzer- und ereignisadaptive, »intelligente« Agenten- und Mediatoren-Software.

7.4.2 Prefetching auf der Basis von Markov-Modellen

Die bisher beschriebenen Prefetching-Verfahren arbeiten kontextfrei in dem Sinne, dass sie die Hitze der Prefetching-Kandidaten unabhängig vom aktuellen Arbeitskontext des Benutzers bzw. der Benutzer (z.B. hinter einem Proxy-Cache) schätzen. Diese Schätzungen beruhen also mathematisch gesehen auf stationärem Verhalten. Wenn man dagegen die nächsten Benutzerschritte als stochastische Funktion des aktuellen Zustands einer Benutzersitzung modelliert, führt dies in natürlicher Weise auf das transiente Verhalten einer *Markov-Kette* [Tijm94]. Unter Sitzung verstehen wir dabei eine Folge von Objektzugriffen, die aus Sicht des Benutzers in einem Kontext stehen, also keinesfalls eine TCP-Sitzung.

Ein solches Modell ist durch eine Menge von Zuständen charakterisiert sowie durch Übergangswahrscheinlichkeiten zwischen Zuständen. Ein Zustand i bedeutet dabei, dass der Benutzer aktuell auf die Webseite i zugreift. Die Markov-Eigenschaft des Modells verlangt, dass die Wahrscheinlichkeit, dass wir im nächsten Schritt in Zustand j sind (d.h. auf Seite j zugreifen), nur vom aktuellen Zustand abhängt und nicht von der Zustandshistorie des so modellierten stochastischen Prozesses. Demzufolge müssen wir anhand von Beobachtungen über das Benutzerverhalten die *Übergangswahrscheinlichkeiten* p_{ij} für Übergange von einem Zustand i in einen Zustand j schätzen. Die Matrix dieser Wahrscheinlichkeiten ist natürlich sehr dünn besetzt; überdies kommt es auch nur darauf an, hinreichend große Transitionswahrscheinlichkeiten zu schätzen. Übergänge von Seite i zu Seite j sind bei einem solchen Modell keineswegs nur auf Seiten beschränkt, die per Hyperlink direkt verbunden sind. Vielmehr spiegeln die Transitionen einfach nur die Korrelation zwischen Seiten in den Zugriffsmustern der Benutzer wider. Beispielsweise sind verschiedene Treffer, die eine Suchmaschine auf einer Resultatseite zurückliefert, korreliert, da der Benutzer mit einer gewissen Wahrscheinlichkeit einen Treffer nach dem anderen anklickt. Die Suchmaschinenseite selbst verbindet gedanklich die Trefferseiten transitiv, allerdings sieht z.B. ein Proxy-Cache typischerweise nur die Zugriffssequenz der Trefferseite, da die Suchmaschi-

nenseite im Client-Cache liegt. Die Allgemeinheit des Modells, das über Hyperlinks hinausgeht, ist also auch im Hinblick auf die Praktikabilität des Sammelns und Aktualisierens von Statistiken essenziell.

Wenn nun eine Benutzersitzung aktuell im Zustand i ist, kann die Prefetching-Software die wahrscheinlichsten Folgezustände direkt an den geschätzten p_{ij}-Werten ablesen und darüber hinaus die Wahrscheinlichkeiten errechnen, dass ein beliebiger Zustand j in den nächsten n Schritten erreicht wird, also auf das entsprechende Objekt zugegriffen wird. Diese Berechnung der *n-Schritt-Wahrscheinlichkeiten* $p_{ij}^{(n)}$ basiert auf der Chapman-Kolmogorov-Gleichung $p_{ij}^{(n)} = \Sigma_k \, p_{ik}^{(n-1)} * p_{kj}$ mit der Verankerung $p_{ij}^{(1)} = p_{ij}$.

Ein auf der Schätzung der n-Schritt-Wahrscheinlichkeiten beruhendes Prefetching-Verfahren könnte nun einfach einen Wert für n fixieren (z.B. n=3), die Wahrscheinlichkeiten berechnen (unter Ausnutzung des Umstands, dass viele dieser Wahrscheinlichkeiten schnell gegen Null gehen) und schließlich alle Objekte als Prefetching-Kandidaten identifizieren, deren n-Schritt-Zugriffswahrscheinlichkeit einen spezifizierten Schwellwert überschreitet. Solche Verfahren sind in der Literatur mehrfach vorgeschlagen worden (siehe z.B. [Best96, PaMo96, FCLJ99]). Um das Vorladen selbst auszulösen, kann man wie im vorhergehenden Unterabschnitt einen weiteren Nutzen-Schaden-Vergleich durchführen und damit das Prefetching konservativ drosseln.

Eine Verallgemeinerung des skizzierten Verfahrens ist der Übergang zu Markov-Ketten höherer Ordnung, bei denen die Wahrscheinlichkeiten der Folgezustände vom aktuellen Zustand und einer konstanten Anzahl vorhergehender Zustände, also einem beschränkten Ausschnitt der Zugriffshistorie, abhängen. Um Prefetching-Kandidaten zu bestimmen, wird das Muster der letzten Zugriffe mit den Mustern der Statistiken verglichen, und es werden die Wahrscheinlichkeiten für die möglichen Fortsetzungen des Musters berechnet. Verfahren dieser Art sind in [PaZd91, CVK93] entwickelt und in [FCLJ99] auf Web Prefetching angewandt worden; Simulationsexperimente zeigen sehr gute Ergebnisse. Bei diesen – wie auch schon bei den einfacheren Verfahren mit Markov-Ketten 1. Ordnung – muss eine gute Implementierung zwangsläufig den für die Statistiken verwendeten Speicherplatz limitieren und mit Heuristiken alte oder wenig signifikante Statistikeinträge kontinuierlich entfernen.

Eine Alternative zu den in den skizzierten Verfahren betrachteten n-Schritt-Wahrscheinlichkeiten besteht darin, die erwartete Anzahl der Zugriffe auf die verschiedenen Objekte für die nächsten n Schritte, ausgehend vom aktuellen Zustand bzw. Suffix der bisherigen Zugriffshistorie, zu berechnen. Dies erfordert höheren Rechenaufwand, kann aber mit Standardmethoden der transienten Analyse von Markov-Ketten durchaus mit akzeptabler Effizienz implementiert werden. Der so ermittelte Wert für ein Objekt x – nennen wir ihn $N_{spec}(x)$ für die spekulativ erwartete Anzahl der Zugriffe auf x – ist gegenüber den bisher betrachtete n-Schritt-Wahrscheinlichkeiten eigentlich der bessere Gradmesser der Wichtigkeit eines Prefetching-Kandidaten. [KrWe97, KrWe98] beschreiben ein auf diesem

Maß beruhendes Verfahren und zeigen dessen Vorteile anhand von Simulationsexperimenten.

Die bisherigen Überlegungen zielen primär darauf ab, zukünftige Zugriffe in *einer* Benutzersitzung vorherzusagen und deren Latenzzeit durch Prefetching zu maskieren. Insofern sind die Verfahren vor allem für die Stufe des Client-Caching interessant. Bei Proxy-Caches oder den verschiedenen Caching-Stufen auf der Server-Seite aber gilt es, die Zugriffe verschiedener Benutzer und die Spekulationen darüber gesamthaft zu berücksichtigen. Beispielsweise kann ein Datenobjekt mit einem relativ kleinen Wert der n-Schritt-Wahrscheinlichkeit oder der erwarteten Zugriffsanzahl N_{spec}, das von vielen Benutzersitzungen gebraucht werden könnte, wichtiger sein als eines mit sehr hohem N_{spec}-Wert einer einzigen Sitzung, dessen Vorladen eben nur für einen Benutzer potenziellen Nutzen bringt. Mit anderen Worten: Die Schätzungen der Wichtigkeit für eine Sitzung müssen über alle gerade aktiven Sitzungen aggregiert werden. Dabei ist jedoch darauf zu achten, dass die Werte verschiedener Sitzungen überhaupt vergleichbar sind. Unsere bisherigen Maße – n-Schritt-Wahrscheinlichkeit und N_{spec} – sind auf den abstrakten Begriff diskreter Schritte in einer Sitzung fixiert. Wenn sich Benutzer unterschiedlich schnell von Webseite zu Webseite bewegen, können die Wichtigkeitsmaße pro Sitzung nicht einfach zusammengefasst werden. Dieses Problem wird am Beispiel von Abbildung 7-2 illustriert. Hier werden die Sitzungen zweier Benutzer betrachtet, die aktuell auf Objekt i bzw. Objekt k zugreifen (in der Abbildung durch den hellgrau gefärbten bzw. schraffierten Zustand hervorgehoben). Die Übergangswahrscheinlichkeiten zu möglichen Objekten für den jeweils nächsten Sitzungszustand seien für beide Sitzungen identisch und durch die Kantenannotationen gegeben (z.B. $p_{if} = 0.8$). Zusätzlich sind die Zustände selbst mit den mittleren Verweilzeiten annotiert (z.B. $H_i = 10$ Sekunden), also der jeweiligen Zeit bis zum Zugriff auf das nächste Objekt in einer Sitzung; diese Verweilzeiten seien für alle Sitzungen identisch und aufgrund von Statistiken geschätzt. Wenn man die Verweilzeiten außer Acht lässt, hat Objekt g die höchste Wahrscheinlichkeit in ein oder zwei Schritten angefordert zu werden, nämlich 0.838 = 0.1 (in einem Schritt von Sitzung 1) + 0.8*0.9 (in zwei Schritten von Sitzung 1) + 0.1 (in einem Schritt von Sitzung 2) – 0.82*0.1 (von beiden Sitzungen), wäre also der beste Prefetching-Kandidat. Berücksichtigt man aber die Verweilzeiten und fragt nach dem besten Prefetching-Kandidaten innerhalb der nächsten 10 Sekunden, sieht man, dass Objekt g sich gar nicht sonderlich zum Vorladen aufdrängt, da der Zugriffspfad, der mit Abstand am meisten zu der hohen Zugriffswahrscheinlichkeit beiträgt, derjenige über Objekt f ist und dort im Mittel 30 Sekunden verweilt wird.

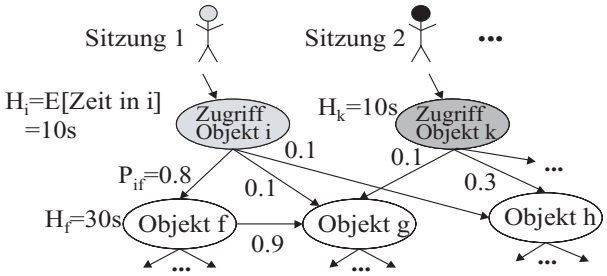


Abb. 7-2: *Beispielszenario für Markov-Modell mehrerer Benutzersitzungen*

Um Szenarien wie dieses angemessen modellieren zu können und um generell mit Caching-Stufen umgehen zu können, die mehrere, gleichzeitige Benutzersitzungen bedienen und Prefetching erwägen, sind die folgenden Aspekte zu berücksichtigen:

- Anstatt diskreter Schritte der Zugriffsmuster sollte die *reale, kontinuierliche, Zeit* modelliert werden, so dass die sitzungsspezifischen Werte von N_{spec} – jetzt auf eine bestimmte Zeitspanne der Realzeit bezogen – vergleichbar werden.
- Die Wichtigkeitsmaße verschiedener Sitzungen müssen in sauberer Weise *aggregiert* werden, um global günstige (oder sogar optimale) Prefetching-Entscheidungen zu treffen.
- Es gilt zu berücksichtigen, dass jederzeit *neue Sitzungen* zu den gerade aktiven hinzukommen können und dass laufende Sitzungen zu verschiedenen Zeitpunkten beendet werden.

Ein diese Aspekte berücksichtigendes, umfassendes Modell zur Vorhersage von Zugriffshäufigkeiten, zur Nutzen-Kosten-Bewertung und zur Identifikation guter Prefetching-Kandidaten wurde in [KrWe97, KrWe98] entwickelt. Dort wird statt eines Markov-Modells mit diskreter Zeit für jede Benutzersitzung eine *Markov-Kette mit kontinuierlicher Zeit* verwendet, und die Vorhersagen für verschiedene Sitzungen werden geeignet überlagert. Das darauf basierende Prefetching-Verfahren *MCMin* (für Markov-Chain-based Data Migration Method) zeigt in Simulationen sehr vielversprechende Resultate.

7.5 Cache-Kohärenz: Wie sollen Daten aktualisiert werden?

Bei jeder Art von verteilter Replikation ergibt sich ein potenzielles *Kohärenzproblem*: Wenn sich ein Datenobjekt ändert, sind Kopien des Objekts, die in anderen Rechnern gehalten werden, veraltet. Ein Kohärenzsteuerungsprotokoll muss diese Situation erkennen, die veralteten Kopien ggf. invalidieren und in diesem Fall dafür sorgen, dass die Rechner, die weiterhin an einer Replikation des modi-

fizierten Objekts interessiert sind, eine aktuelle Kopie erhalten bzw. ihre Kopie aktualisieren können.

Im allgemeinen Fall, dass Änderungen auf jeder beliebigen Kopie initiiert werden können, sind solche Protokolle relativ komplex; sie finden beispielsweise bei Datenbanksystemen in Rechner-Clustern Verwendung und bei verteilten Dateisystemen [Rahm94, WeVo01]. Die in der Praxis bewährten Verfahren sind Eigentümer-basiert: Jedes Datenobjekt hat *genau einen* Rechner als Heimat und kann von dort an andere Rechner ausgeliehen werden. Das Ausleihen impliziert ein Recht, lokale Zugriffe ohne erneute Kommunikation mit dem Heimatrechner autonom durchzuführen. Solange das Objekt nur gelesen wird, kann es von mehreren Rechnern gleichzeitig ausgeliehen, sprich repliziert, werden; diese Rechner sind die aktuellen Eigentümer des Objekts. Sobald das Objekt von einem der Eigentümer geändert werden soll, werden die Kopien aller anderen Eigentümer zurückgerufen und der ändernde Rechner wird zum einzigen Eigentümer. An einen solchen »*Callback*« sind bei Datenbanksystemen globale Synchronisationsmaßnahmen, z.B. Sperrprotokolle, gekoppelt, die die Konsistenz zwischen verschiedenen Datenobjekten sicherstellen.

Beim Web Caching ist die Situation insofern wesentlich einfacher, als dass ein Datenobjekt ohnehin nur von seinem Heimat-Server geändert werden kann. Eine Ausnahme ist das verteilte Caching innerhalb eines Server-Komplexes, also zwischen verschiedenen Applikations- oder Daten-Server-Rechnern desselben Web-Informationsdienstes. In diesem Fall würde man eben mit einem Eigentümer-basierten Protokoll der oben skizzierten Art arbeiten. Entlang der Caching-Hierarchie von der Server-Seite bis zu den Clients kommen Änderungen aber nur auf der Server-Seite vor. Der dadurch möglichen Simplifizierung steht jedoch die zusätzliche Komplexität der Skalierbarkeit gegenüber. Wenn Tausende und mehr Rechner Kopien eines Objekts halten und diese Rechner beliebig im Web verteilt sind, scheiden naive Lösungen wie z.B. Aktualisierung durch Broadcasting schlichtweg aus.

Die in HTTP vorgesehene Standardlösung für das Kohärenzproblem entlang der Caching-Hierarchie beruht darauf, dass Clients mit der HTTP-Methode *Get-If-Modified-Since* ein Objekt beim Heimat-Server konditional anfordern können. Wenn sich das Objekt seit dem beim Aufruf als Parameter mitgelieferten Zeitpunkt verändert hat, wird eine aktuelle Kopie vom Server gesendet; andernfalls wird nur eine kurze Bestätigung geschickt, dass die Kopie des Objekts weiterhin gültig ist.

Selbst diese kurzen Bestätigungsnachrichten beim Aktualitätstest können jedoch eine erhebliche Zusatzlast für das Netzwerk und die Server darstellen. Umgekehrt können die meisten Web-Anwendungen aber durchaus mit u.U. geringfügig veralteten Informationen leben; es genügt z.B. meist, dass Produkt- oder Fahrplaninformationen nicht älter als einen Tag sind, und Daten aus digitalen Bibliotheken oder Tourismusinformationssystemen sind diesbezüglich noch unkritischer. Im Zweifelsfall kann der Benutzer durch entsprechende Einstellung seines Browsers oder explizites Klicken der Refresh-Funktion den Zugriff auf den

Heimat-Server erzwingen, doch wird davon nur sehr selten Gebrauch gemacht. In Anbetracht dieser entspannteren Anwendungsanforderungen und im Hinblick auf das Skalierbarkeitsproblem hat das HTTP-Protokoll vorgesehen, dass Server beim Ausliefern eines Datenobjekts dieses mit einer so genannten *Time-To-Live (TTL)* versehen können. Der TTL-Wert ist ein Schätzwert für die Zeit bis zur nächsten Änderung des Objekts durch den Heimat-Server; in vielen Fällen wird dies in der Größenordnung von Tagen liegen. Das Standardverhalten der verschiedenen Caching-Stufen ist nun, selbst auf ein Get-If-Modified-Since zu verzichten, solange die TTL-Zeit eines replizierten Objekts (seit der Einlagerung in den Cache) noch nicht abgelaufen ist. Zu beachten ist dabei allerdings, dass die TTL-Werte nur vom Heimat-Server gesetzt werden. Wenn ein Cache das Objekt von einem anderen, näher am Heimat-Server liegenden Cache erhält, muss die in dem anderen Cache seit der dortigen Einlagerung verbrachte Zeit mitberücksichtigt werden. Dieser Effekt ist transitiv und hat durchaus Auswirkungen auf die Cache-Effektivität [CoKa01].

Ein Cache erkennt also die Nichtaktualität seiner Kopie eines Datenobjekts in der Regel erst bei Ablauf der TTL-Frist des Objekts; dann wird eine Get-If-Modified-Since-Nachricht fällig. Natürlich kann im Sinne eines Prefetching-Verfahrens die konditionale Anforderung auch schon vorzeitig geschickt werden (Prevalidation, Rejuvenation), aber dies wäre einzig in der Verantwortung der jeweiligen Caching-Stufe. Eine solche Architektur, bei der sich nur die Clients bzw. näher beim Client liegenden Caching-Stufen aktiv um die Aktualisierung ihrer replizierten Daten bemühen, wird auch *Pull-basiert* genannt. Dem stehen so genannte *Push-basierte* Architekturen gegenüber, bei denen die Server-Seite sich aktiv um die Aktualisierung ihrer ausgeliehenen Datenobjekte kümmert. Im Extremfall abonnieren Clients einfach einen Datenstrom (Data Feed), den sie dann vom jeweiligen Server unaufgefordert erhalten, und zwar zu Zeitpunkten, die einzig der Server bestimmt. Dieser Extremfall einer Push-Architektur war in der Vergangenheit (z.B. Pointcast) aber wenig erfolgreich. Heute scheinen sich eher kombinierte *Push/Pull-Architekturen* und so genannte *Publish-Subscribe-Architekturen* (im Jargon der IT-Branche auch kurz »PubSub« genannt) durchzusetzen [AFZ97, Fran00, Deol01], beispielsweise für das Abonnieren von Nachrichten-Feeds, Börsenkursen, u.ä.

Der Server-Einfluss in solchen Architekturen kann unterschiedlich stark ausfallen. Im einfachsten Fall schickt ein Server mit einer beliebigen, ohnehin zu sendenden Nachricht an einen Cache, z.B. einen Proxy, im »Huckepack-Verfahren« (Piggybacking) eine Liste der Bezeichner (z.B. URLs) von Objekten, die der Cache vor nicht allzu langer Zeit angefordert hatte und die vom Server seitdem modifiziert wurden [KrWi99]. Auf diese Weise erfährt der Cache viel früher, dass bestimmte seiner replizierten Objekte nicht mehr aktuell sind, und kann dann weitere Maßnahmen ergreifen. Die Zusatzbelastung für das Netzwerk ist akzeptabel, da zwar mehr Daten verschickt werden, aber keine zusätzlichen TCP-Verbindungen benötigt werden.

Eine stärkere Rolle spielt der serverseitige Push-Aspekt in den Verfahren von [YBS99, YADI01]. Hier wird das Caching eines Objekts wie bei den eingangs dieses Abschnitts angesprochenen Datenbanktechniken als Ausleihen vom Heimat-Server des Objekts interpretiert. Der Server führt (ggf. approximativ) Buch darüber, welche Caches welche Objekte repliziert haben, und informiert aktiv die betroffenen Caches, wenn sich Objekte geändert haben.

Insgesamt steht das Spektrum der Möglichkeiten im Spannungsfeld zwischen Kohärenz und Skalierbarkeit: Perfekte Kohärenz und Konsistenz von Daten sind nicht ohne weiteres auf die Größenordnung des Web skalierbar, und die Kunst besteht darin, skalierbare Verfahren zu realisieren, die nur geringe Abstriche bezüglich der Kohärenz machen und die Anforderungen der entsprechenden Web-Anwendungen erfüllen können. Leistungsuntersuchungen zu diesen Aspekten der diskutierten Pull-/Push-Kombinationsverfahren finden sich u.a. in [AFZ97, KrWi99, Deol01].

7.6 Zusammenfassung und Ausblick

In diesem Kapitel haben wir einen Überblick über Architekturen, Mechanismen und Strategien für Web Caching gegeben. Wir haben gesehen, dass es vielfältige Möglichkeiten für hierarchisches Caching entlang des Pfades vom Client zum Heimatrechner eines Datenobjekts gibt; darüber hinaus ist der Einsatz von verteiltem, kooperativem Caching zwischen verschiedenen Edge-Server-Rechnern oder verschiedenen HTTP-, Applikations- oder Daten-Server-Rechnern desselben Web-Informationsanbieters möglich und in vielen Fällen sinnvoll. Caching kann nicht nur die Zugriffslatenzzeit auf statische HTML-Seiten und eingebettete Objekte drastisch verringern, sondern auch dynamische Webseiten und Datenbankanfrageresultate materialisieren und replizieren und damit moderne Web-Anwendungen signifikant beschleunigen.

Die bereitzustellenden Mechanismen und Strategien umfassen die Cache-Ersetzungsverfahren und die Aktualisierung replizierter Objekte sowie – optional, aber zur Maskierung von Latenzzeiten vielfach leistungssteigernd – das Prefetching von Objekten. Ersetzungs- und Prefetching-Strategien sollten sowohl die aktuelle Wichtigkeit (Popularität) von Objekten berücksichtigen als auch deren Größe und die – in der Regel heterogenen – Zugriffskosten auf entfernte Objekte. Aus diesen Gründen sind generische Standardstrategien wie LRU zwar im Web Caching einsetzbar, leistungsmäßig den nutzenorientierten Verfahren aber deutlich unterlegen. Wichtigkeit kann dabei aufgrund relativ einfacher, stationärer Statistiken geschätzt werden, z.B. mit dem LRU-k-Verfahren, oder – ambitionierter, potenziell besser, aber auch aufwändiger – mit kontextbewussten Techniken, die auf Markov-Ketten beruhen.

Bei der Betrachtung von Aktualisierungsverfahren haben wir gesehen, dass die Grenzen zwischen Client-gesteuerten, so genannten Pull-Methoden, und Server-gesteuerten Push-Methoden verschwimmen. In *Content-Delivery-Netzen (CDNs)*

wie z.B. Akamai [Karg99, GCR00] verschwimmen sogar die Grenzen zwischen temporärem Caching und langfristig optimierter Datenplatzierung und –replikation, da die Verteilung der Daten über Proxies, Edge-Server und Heimat-Server dynamisch und lastadaptiv geändert werden kann [FCAB98]. In *Peer-to-Peer-Anwendungen* wie z.B. dem beliebten File-Sharing-Dienst Gnutella (und früher Napster) schließlich verschwindet die Unterscheidung zwischen Clients, Proxies und Server-Rechnern gänzlich (siehe z.B. [RoDr01]).

Die Konsequenz aus diesen modernen Trends ist, dass man künftig Datenplatzierung, Caching und Prefetching noch umfassender und integrierter optimieren muss und zudem in noch stärkerem Maße auf intelligente, lastadaptive und selbstoptimierende Strategien setzen wird, die selbst in dynamisch variablen Netzstrukturen mit drahtloser Kommunikation und hochgradig mobilen Endgeräten weitgehend ohne Systemadministrator auskommen. Die Herausforderungen dieser Art von »Ubiquitous Data Access« [Fran00] werden Forschung und Entwicklung noch etliche Jahre beschäftigen. Dabei muss angestrebt werden, statt der jetzt vorherrschenden »Best-Effort«-Mentalität des Internets umfassendere und rigorose *Quality-of-Service-Garantien* bezüglich Datenverfügbarkeit und Antwortzeiten zu realisieren [Weik00]. Offensichtlich erfordert dies nicht nur intelligentere Strategien und Optimierungen in der Infrastruktur des Web, sondern auch eine raffiniertere Differenzierung zwischen den Anforderungen und den zu investierenden Ressourcen verschiedener Benutzer- und Dienstkategorien. Hier gibt es Anknüpfungspunkte zu früheren Arbeiten über *zielorientiertes automatisches Tuning*, insbesondere im Bereich von OLTP und anderen einfachen Datenbankanwendungen [FGND93, Rahm97]. Durch die enorme Größenordnung und Komplexität des Web sind praktikable Lösungen aber bislang nicht in Reichweite; die langfristig angestrebte Form von »*Differentiated Quality of Service*« auf der Ebene der E-Services und sonstige Web-Anwendungen (siehe z.B. [KSWD01]) ist ein aktuelles, herausforderndes und spannendes Forschungsthema.

Literatur

[AFZ97] Acharya, S., Franklin, M.J., Zdonik, S.B., *Balancing Push and Pull for Data Broadcast*, ACM SIGMOD Conference, 1997.

[Anto02] Anton, J., Jacobs, L., Liu, X., Parker, J., Zeng, Z., Zhong, T., *Web Caching for Database Applications with Oracle Web Cache*, ACM SIGMOD Conference, 2002.

[BBK00] Bhatti, N., Bouch, A., Kuchinsky, A., *Integrating User Perceived Quality into Web Server Design*, 9th WWW Conference, 2000.

[Best96] Bestavros, A., *Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time in Distributed Information Systems*, IEEE ICDE Conference, 1996.

[CABK88] Copeland, G., Alexander, W., Boughter, E., and Keller, T., *Data Placement in Bubba*, ACM SIGMOD Conference, 1988.

[CaIr97] Cao, P., Irani, S., *Cost-aware WWW Proxy Caching Algorithms*, USENIX Symposium on Internet Technologies and Systems, 1997.

[CFKL95] Cao, P., Felten, E.W., Karlin, A.R., Li, K., *A Study of Integrated Prefetching and Caching Strategies*, ACM SIGMETRICS Conference, 1995.

[ChHa80] Chandra, A.K., Harel, D., *Structure and Complexity of Relational Queries*, IEEE Symposium on Foundations of Computer Science, 1980.

[CID99] Challenger, J., Iyengar, A., Dantzig, P., *A Scalable System for Consistently Caching Dynamic Web Data*, IEEE INFOCOM Conference, 1999.

[CoDe73] Coffman, E., Denning, P., *Operating Systems Theory*, Prentice-Hall, 1973.

[CoKa99] Cohen, E., Kaplan, H., *Caching Documents with Variable Sizes and Fetching Costs: An LP-Based Approach*, ACM-SIAM Symposium on Discrete Algorithms, 1999.

[CoKa01] Cohen, E., Kaplan, H., *Refreshment Policies for Web Content Caches*, IEEE INFOCOM Conference, 2001.

[CrBa98] Crovella, M., Barford, P., *The Network Effects of Prefetching*, IEEE INFOCOM Conference, 1998.

[CVK93] Curewitz, K.M., Krishnan, P., J.S. Vitter, *Practical Prefetching Via Data Compression*, ACM SIGMOD Conference, 1993.

[Dar96] Dar, S., Franklin, M.J., Jonsson, B.T., Srivastava, D., Tan, M., *Semantic Data Caching and Replacement*, International Conference on Very Large Data Bases, 1996.

[Deol01] Deolasee, P., Katkar, A., Panchbudhe, A., Ramamritham, K., Shenoy, P., *Adaptive Push-Pull: Disseminating Dynamic Web Data*, WWW Conference, 2001.

[DILR00] Degenaro, L., Iyengar, A., Lipkind, I., Rouvellu, I., *A Middleware System Which Intelligently Caches Query Results*, International Conference on Distributed Systems Platforms (Middleware 2000), 2000.

[DWAP94] Dahlin, M.D., Wang, R.Y., Anderson, T.E., Patterson, D.A., *Cooperative Caching: Using Remote Client Memory to Improve File System Performance*, 1st International Symposium on Operating Systems Design and Implementation, 1994.

[FCAB98] Fan, L., Cao, P., Almeida, J., Broder, A.Z., *Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol*, ACM SIGCOMM Conference, 1998.

[FCLJ99] Fan, L., Cao, P., Lin, W., Jacobson, Q., *Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance*, ACM SIGMETRICS Conference, 1999.

[FGND93] Ferguson, D., Georgiadis, L., Nikolaou, C., Davies, K., *Satisfying Response Time Goals in Transaction Processing Systems*, International Conference on Parallel and Distributed Information Systems (PDIS), 1993.

[Fran00] Franklin, M.J., *Challenges in Ubiquitous Data Management*, in: R. Wilhelm (Editor), Informatics – 10 Years Back, 10 Years Ahead, LNCS 2000, Springer, 2000.

[GCR00] Gadde, S., Chase, J., Rabinovich, M., *Web Caching and Content Distribution: A View From the Interior*, 5th Int. Web Caching and Content Delivery Workshop (WCW'00), May 2000.

[GrSh00] Gray, J., Shenoy, P.J., *Rules of Thumb in Data Engineering*, IEEE International Conference on Data Engineering, 2000.

[HäRa01] Härder, T., Rahm, E., *Datenbanksysteme: Konzepte und Techniken der Implementierung*, Springer, 2001.

[JiBe00] Jin, S., Bestavros, A., *Popularity-aware GreedyDual-Size Web Proxy Caching Algorithms*, IEEE International Conference on Distributed Computing Systems, 2000.

[JiKl98] Jiang, Z., Kleinrock, L., *An Adaptive Network Prefetch Scheme*, IEEE Journal of Selected Areas in Communications Vol.16 No.3, 1998.

[Karg99] Karger, D.R. et al., *Web Caching with Consistent Hashing*. WWW Conference, 1999.

[KrRe01] Krishnamurthy, B., Rexford, J., *Web Protocols and Practice*, Addison-Wesley, 2001.

[KrWe97] Kraiss, A., Weikum, G., *Vertical Data Migration in Large Near-Line Document Archives Based on Markov-Chain Predictions*, VLDB Conference, 1997.

[KrWe98] Kraiss, A., Weikum, G., *Integrated Document Caching and Prefetching in Storage Hierarchies Based on Markov-Chain Predictions*, VLDB Journal Vol.7 No.3, 1998.

[KrWi99] Krishnamurthy, B., Wills, C.E., *Proxy Cache Coherence and Replacement – Towards a More Complete Picture*, IEEE International Conference on Distributed Computing Systems, 1999.

[KSWD01] Kraiss, A., Schoen, F., Weikum, G., Deppisch, U., *Towards Response Time Guarantees for e-Service Middleware*, IEEE CS Data Engineering Bulletin Vol.24 No.1, 2001.

[LaYa85] Larson, P.-A., Yang, H.Z., *Computing Queries from Derived Relations*, VLDB Conference, 1985.

[Lee99] Lee, D., Choi, J., Kim, J.-H., Noh, S.H., Min, S.L., Cho, Y., Kim, C.S., *On the Existence of a Spectrum of Policies that Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies*, ACM SIGMETRICS Conference, 1999.

[LGS00] Loosley, C., Gimarc, R.L., Spellmann, A.C., *E-Commerce Response Time: a Reference Model*, Keynote Systems Inc., 2000, http://www.keynote.com/ services/assets/ applets/ECommerce_Response_Time_CMG_2000__Chris_.pdf

[LRV96] Lorenzetti, P., Rizzo, L., Vicisano, L., *Replacement Policies for a Proxy Cache*, Technical Report, Universita di Pisa, Italy, 1996.

[LuNa01] Luo, Q. Naughton, J.F., *Form-based Proxy Caching for Database-backed Web Sites*, VLDB Conference, 2001.

[Luo02] Luo, Q., Krishnamurthy, S., Mohan, C., Pirahesh, H., Woo, H., Lindsay, B.G., Naughton, J.F., *Middle-Tier Database Caching for e-Business*, ACM SIGMOD Conference, 2002.

[LWY96] Leff, A., Wolf, J.L., Yu, P.S., *Efficient LRU-based Buffering in a LAN Remote Caching Architecture*, IEEE Transactions on Parallel and Distributed Systems Vol.7 No.2, 1996.

[Moha01] Mohan, C., *Caching Technologies for Web Applications*, Tutorial Notes, VLDB Conference, 2001.

[OOW93] O'Neil, E.J., O'Neil, P.E., Weikum, G., *The LRU-K Page Replacement Algorithm for Database Disk Buffering*, ACM SIGMOD Conference, 1993.

[OOW99] O'Neil, E.J., O'Neil, P.E., Weikum, G., *An Optimality Proof of the LRU-K Page Replacement Algorithm*, Journal of the ACM Vol.46 No.1, 1999.

[PaMo96] Padmanabhan, V., Mogul, J.C., *Using Predictive Prefetching to Improve World Wide Web Latency*, ACM SIGCOMM Conference, 1996.

[Patt95] Patterson, R.H., Gibson, G.A., Ginting, E., Stodolsky, D., Zelenka, J., *Informed Prefetching and Caching*, ACM Symposium on Principles of Operating Systems, 1995.

[Rahm94] Rahm, E., *Mehrrechner-Datenbanksysteme*, Addison-Wesley, 1994.

[Rahm97] Rahm, E., *Goal-oriented Performance Control for Transaction Processing*, German Conference on Performance Assessment, 1997.

[RaSp02] Rabinovich, M., Spatscheck, O., *Web Caching and Replication*, Addison-Wesley, 2002.

[RoDr01] Rowstron, A., Druschel, P., *Storage Management and Caching in PAST: a Large-scale, Persistent, Peer-to-peer Storage Utility*, ACM Symposium on Operating System Principles, 2001.

[SiWe97] Sinnwell, M., Weikum, G., *A Cost-Model-Based Online Method for Distributed Caching*, IEEE ICDE Conference, 1997.

[SILD02] Song, J., Iyengar, A., Levy-Abegnoli, E., Dias, D., *Architecture of a Web Server Accelerator*, Computer Networks Vol.38 No.1, 2002.

[SSV96] Scheuermann, P., Shim, J., Vingralek, R., *WATCHMAN: A Data Warehouse Intelligent Cache Manager*, VLDB Conference, 1996.

[SSV97] Scheuermann, P., Shim, J., Vingralek, R., *A Case for Delay-Conscious Caching of Web Documents*, WWW Conference, 1997.

[SSV99] Shim, J., Scheuermann, P., Vingralek, R., *Proxy Cache Design: Algorithms, Implementation, and Performance*, IEEE Transactions on Knowledge and Data Engineering, 1999.

[Tijm94] Tijms, H.C., *Stochastic Models: An Algorithmic Approach*, John Wiley & Sons, 1994.

[VLN98] Venkataraman, S., Livny, M., Naughton, J.F., *Remote Load-Sensitive Caching for Multi-Server Database Systems*, IEEE International Conference on Data Engineering, 1998.

[Weik00] Weikum, G., *The Web in 10 Years: Challenges and Opportunities for Database Research*, in: R. Wilhelm (Editor), Informatics – 10 Years Back, 10 Years Ahead, LNCS 2000, Springer, 2000.

[WeVo01] Weikum, G., Vossen, G., *Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery*, Morgan Kaufmann, 2001.

[Wolm99] Wolman, A., Voelker, G.M. Sharma, N., Cardwell, N. Karlin, A., Levy, H.M., *On the Scale and Performance of Cooperative Web Proxy Caching*, ACM Symposium on Operating System Principles, 1999.

[YADI01] Yin, J., Alvisi, L., Dahlin, M., Iyengar, A., *Engineering Server-driven Consistency for Large Scale Dynamic Web Services*, WWW Conference, 2001.

[YBS99] Yu, H., Breslau, L., Shenker, S., *A Scalable Web Cache Consistency Architecture*, ACM SIGCOMM Conference, 1999.

[YFIV00] Yagoub, K., Florescu, D., Issarny, V., Valduriez, P., *Caching Strategies for Data-intensive Web Sites*, VLDB Conference, 2000.

[YWM00] Yang, J., Wang, W., Muntz, R.R., *Collaborative Web Caching Based on Proxy Affinities*, ACM SIGMETRICS Conference, 2000.

[Young94] Young, N., *The k-Server Dual and Loose Competitiveness for Paging*, Algorithmica Vol.11 No.6, 1994.

8 Indexstrukturen für XML

Sven Helmer, Guido Moerkotte

Kurzfassung

Indexstrukturen sind ein wichtiges Mittel, um die Antwortzeiten eines Datenbanksystems zu verbessern. Neue Anwendungen wie die Verarbeitung von XML-Dokumenten stellen Anforderungen, die von traditionellen Indexstrukturen nicht erfüllt werden. Wir skizzieren zunächst die Anforderungen der XML-Verarbeitung in Form von verschiedenen Anfragearten. Dabei spielt die Auswertung von Pfadausdrücken eine zentrale Rolle. Im Hauptteil des Kapitels beschreiben wir eine Reihe von Indexstrukturen, die entweder Weiterentwicklungen bereits existierender Indexstrukturen darstellen oder Neuentwicklungen sind, und betrachten sie hinsichtlich der Unterstützung der diversen Anfragearten. Eine kurze Evaluierung der einzelnen Ansätze rundet das Kapitel ab.

8.1 Einleitung und Motivation

Ein Ansatz, die Antwortzeiten eines Datenbanksystems zu verbessern, besteht darin, möglichst nicht den gesamten Datenbestand bei einer Anfragebearbeitung zu durchsuchen, da dies sehr ineffizient ist. Eine umfassende Suche ist üblicherweise auch nicht nötig, da meistens nur ein kleiner Prozentsatz aller Daten die Kriterien der Anfrage erfüllt. Deswegen sollte ein System in der Lage sein, auf die für eine Anfrage relevanten Daten direkt zuzugreifen. Datenstrukturen, die diesen schnellen Zugriff erlauben, werden Indexstrukturen genannt.

Neue Anwendungen, wie z.B. die Verarbeitung von XML-Dokumenten, stellen Anforderungen an Datenbanksysteme, die weit über diejenigen klassischer Anwendungen, wie z.B. Verwaltung buchhalterischer Daten, hinausgehen. Traditionelle Indexstrukturen, wie z.B. B-Bäume oder invertierte Listen, sind mit diesen Anforderungen in der Regel überfordert. Wir werden zunächst verschiedene Anfragetypen auf XML-Dokumenten klassifizieren und danach skizzieren, welche Indexstrukturen die jeweiligen Anfragen unterstützen.

Ein wichtiger Anfragetyp, der auch bei XML-Dokumenten weiterhin eine Rolle spielen wird, ist die *Volltextanfrage*. Hier geht es um das Auffinden von Wortvorkommen in Dokumenten. Dabei kann auf die umfangreichen Erfahrungen mit Volltextindexen aus dem Bereich des Information Retrieval zurückgegrif-

fen werden. Diese Indexstrukturen erlauben eine schnelle Textsuche in Dokumenten, ignorieren dabei aber deren Struktur.

Ein weiterer Anfragetyp ist die *Werteanfrage*, bei der es um das Finden von atomaren Werten in abgespeicherten Datenobjekten geht. Dies ist die klassische Domäne von Datenbanksystemen, in der mit sehr strukturierten Daten gearbeitet wird. Datenobjekte werden mit Attributen beschrieben, auf deren Werte später in Anfragen Bezug genommen wird.

Navigierende Anfragen, die sich an der Struktur der Daten orientieren, sind bereits aus dem Bereich der Netzwerk- und objektorientierten Datenbanken bekannt. Neu im Zusammenhang mit XML-Dokumenten ist dabei, dass die Struktur nicht mehr fest vorgegeben ist, d.h. bei der Bearbeitung einer navigierenden Anfrage nicht mehr unbedingt von einem bekannten Schema ausgegangen werden kann.

Die Navigation in einem XML-Dokument wird durch *Pfadausdrücke* (z.B. in XPath) beschrieben. Im Hinblick auf Indexstrukturen sind dabei verschiedene Varianten interessant, deren Handhabung unterschiedlich schwierig ist. Am einfachsten sind *Wurzelpfadanfragen*, also abstrakte Pfade, die die Wurzel eines Dokuments einbeziehen. *Kontextpfadanfragen* arbeiten mit Pfaden, die relativ zu beliebigen Kontextknoten im Dokument formuliert werden können. Dies erhöht die Flexibilität, aber zugleich auch die Komplexität der Anfragen. Orthogonal zu diesen beiden Konzepten können *Platzhalter* (Wildcards) in den Pfadausdrücken eingeführt werden, was die Flexibilität und Komplexität noch weiter steigert.

Denkbar sind außerdem *kombinierte Anfragen*, die sich sowohl auf den Inhalt als auch die Struktur von XML-Dokumenten beziehen. Der Übergang zwischen reinen Volltextanfragen und reinen navigierenden Anfragen gestaltet sich fließend.

Eine Klasse für sich bilden die *Tree-Matching*-Anfragen. Während bisher davon ausgegangen wurde, dass die Strukturbeschreibungen in Anfragen durch Pfadausdrücke formuliert werden, wird bei diesem Ansatz direkt nach Teilbäumen (tree patterns) in den Dokumenten gesucht. Man beachte, dass in XPath Teilbäume spezifiziert werden können.

Die wesentlichen Neuerungen bei XML-Indexstrukturen sind im Bereich der Unterstützung von Anfragen anzusiedeln, die sich zumindest teilweise auf die Struktur der Dokumente beziehen. Deswegen konzentrieren wir uns hauptsächlich auf die geeignete Darstellung und Speicherung von Pfaden. In diesem Bereich gab es in neuerer Zeit einige Entwicklungen, die wir hier vorstellen werden.

Bei der Klassifizierung der Indexstrukturen haben wir uns vom Kriterium der Entstehungsweise leiten lassen, ohne jedoch die erwähnten Anfragetypen außer Acht zu lassen. Unserer Meinung nach ist die Funktionsweise leichter zu verstehen, wenn klar ist, aus welchem Umfeld die Indexstrukturen stammen. Dabei teilen wir die Indexstrukturen grob in drei Bereiche ein. Zum einen existieren Ansätze basierend auf Information-Retrieval-Techniken wie Signaturen, invertierte Listen und Patricia Tries. Dazu zählen Kontextfilter, die Techniken im Niagara-System und Index Fabrics. Diese Methoden sind Inhalt des Abschnitts 8.3.

Weitere Ansätze basieren auf traditionellen Datenbanktechniken wie z.B. B-Bäumen. Darunter fallen so genannte Extended Access Support Relations und multidimensionale Indexierung, die wir in Abschnitt 8.4 erläutern. Zu guter Letzt existieren noch Ansätze, die keinem dieser beiden Bereiche zuzuordnen sind. Wir besprechen in Abschnitt 8.5 Data Guides, den T-Index und Tree-Matching.

8.2 Voraussetzungen und Erläuterungen

Semistrukturierte Daten wie XML-Dokumente werden oft als Baum dargestellt. Die Knoten speichern dabei die Namen von Elementen (Tags) bzw. Textinformation. Als Beispiel einer Baumrepräsentation eines XML-Dokuments siehe Abb. 8-1 und Abb. 8-2. Ein Pfad von einem Knoten zu einem anderen wird dabei durch die Namen der Knoten repräsentiert, durch die der Pfad läuft.

```
< ? xml version=«1.0« ? >
<Buch>
        <Autor>
                <Name> Ray </Name>
                <Vorname> Erik T. </Vorname>
        </Autor>
        <Titel> Einführung in XML </Titel>
        <Text>
                <Kapitel>
                        <Titel> Einleitung </Titel>
                        <Abschnitt> Was ist XML? ...</Abschnitt>
                </Kapitel>
                <Kapitel>
                        <Titel> Markup-Konzepte </Titel>
                        <Abschnitt> Die Anatomie ...</Abschnitt>
                        <Abschnitt> Elemente ...</Abschnitt>
                </Kapitel>
        </Text>
</Buch>
```

Abb. 8-1: *Beispiel für ein XML-Dokument*

Das Vorkommen von IDREFs in einem XML-Dokument kann dazu führen, dass eine streng hierarchische Darstellung nicht möglich ist. Viele der vorgestellten Indexstrukturen kommen jedoch auch mit allgemeineren Graphen zurecht, wenn auch mit zusätzlichem Aufwand.

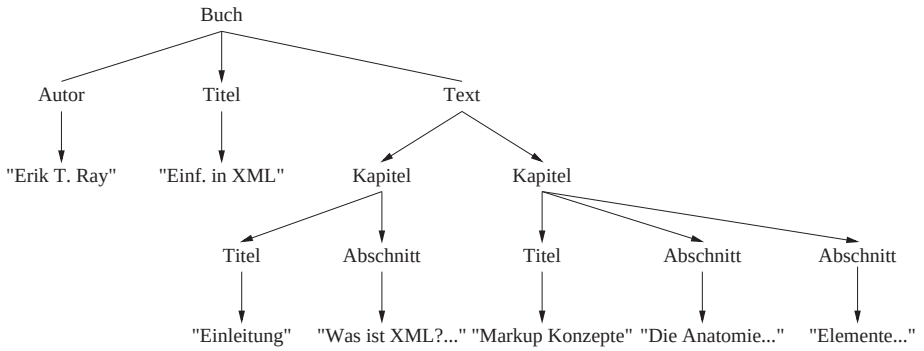


Abb. 8-2: *Beispiel für die Darstellung als Graph*

Üblicherweise werden Strukturbeschreibungen in Anfragesprachen durch Pfadausdrücke repräsentiert. Die geeignete Darstellung und Speicherung von Pfaden ist deswegen ein Hauptanliegen von Indexstrukturen auf semistrukturierten Daten. Für die Unterstützung wertebasierter Vergleiche auf textuellen Knoteninhalten werden oft herkömmliche Volltextindexe wie invertierte Listen eingesetzt. Die wesentlichen Neuerungen sind bei der Indexierung von Pfaden zu finden. Deswegen konzentrieren wir uns hauptsächlich auf diesen Bereich.

8.3 Information-Retrieval-Techniken und Erweiterungen

In diesem Abschnitt besprechen wir die Techniken aus dem Bereich Information Retrieval und darauf basierende Erweiterungen. Zuerst wird für jede Zugriffsmethode kurz erläutert, welche Anfragetypen jeweils unterstützt werden, danach folgt eine kurze Beschreibung der eigentlichen Indexstruktur. Wir beginnen mit traditionellen Indexstrukturen aus dem Bereich Information Retrieval wie Signaturen, invertierte Listen und Tries. Danach werden für XML-Dokumente erweiterte Techniken vorgestellt, nämlich Kontextfilter, SEQL und Index Fabrics.

8.3.1 Signaturen

Dieses Verfahren ist ein Filter für reine Volltextanfragen, d.h., es wird eine Obermenge aller Dokumente bzw. Dokumentteile zurückgeliefert, in denen ein Suchterm auftritt. Diese Antwortmenge muss danach mit herkömmlichen String-Matching-Algorithmen durchsucht werden.

Signaturdateien sind Hash-basierte Indexstrukturen, die Stichwörter (oder andere Informationseinheiten) auf Bitvektoren (so genannte Signaturen) abbilden (für eine ausführlichere Beschreibung siehe [BaRi99]). Die bitweise mit einer Oder-Verknüpfung verbundenen Wortsignaturen eines Dokuments bilden die Signatur des Dokuments. In einer Signaturdatei werden die Dokumentsignaturen zusammen mit einer Referenz auf das jeweilige Dokument abgelegt.

Die Hauptidee dabei ist, zu den Schlüsselwörtern einer Anfrage ebenfalls eine Signatur zu generieren und mit dieser Anfragesignatur die Signaturdatei zu durchsuchen. Der Vergleich von Dokument- und Anfragesignaturen kann dabei mit sehr schnellen Bit-Operationen realisiert werden. Sobald ein Bit gefunden wird, das in der Anfragesignatur gesetzt ist, in einer Dokumentsignatur jedoch nicht, kann dieses Dokument verworfen werden, da unmöglich alle Suchbegriffe darin vorkommen können. Es kann allerdings durch die verlustbehaftete Schlüsseltransformation (hashing) vorkommen, dass alle entsprechenden Bits eines Wortes in einer Dokumentsignatur gesetzt sind, das Wort selbst allerdings nicht im Dokument vorkommt (dieser Fall wird mit »false drop« bezeichnet). Aus diesem Grund müssen die von einer Signaturdatei zurückgelieferten Antworten noch einmal überprüft werden. Durch die Vorfilterung rechnet man aber damit, dass im Allgemeinen die zu prüfende Antwortmenge relativ klein ist.

8.3.2 Invertierte Listen

Bei invertierten Listen (auch invertierte Dateien genannt) handelt es sich ebenfalls um eine Indexstruktur für Volltext-Retrieval. Im Gegensatz zu Signaturen werden hier aber exakte Ergebnisse zurückgeliefert.

Eine invertierte Liste ist vergleichbar mit einem Index, wie er in herkömmlichen Büchern verwendet wird. Sie besteht aus zwei Teilen, einem Verzeichnis, in dem alle Stichwörter gespeichert werden, und den Listen selbst, in denen für jedes Stichwort die Vorkommen verzeichnet sind (siehe Abb. 8-3, die Zahlen in den Listen stehen dabei für Dokumentreferenzen in Form von Dokument-IDs).

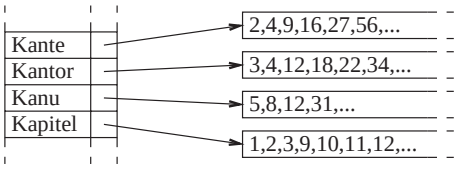


Abb. 8-3: *Invertierte Liste*

Bei Anfragen wird für jeden Suchterm die entsprechende Liste geholt, und je nach (boolescher) Verknüpfung der Suchterme werden die Listen zusammengeführt. Bei einer Oder-Verknüpfung werden die Listen vereinigt, bei einer Und-Verknüpfung miteinander geschnitten. Diese beiden Operationen werden beschleunigt, indem man die Listen sortiert hält.

In einer invertierten Liste können noch mehr Informationen zu einem Suchterm abgelegt werden als lediglich eine Dokumentreferenz. So ist es z.B. sinnvoll, die Positionen eines Stichworts innerhalb eines Dokuments anzugeben. Auf diese Weise können auch Anfragen mit Entfernungsangaben zwischen Wörtern ausgewertet werden. Die Verbesserung invertierter Listen per Anreicherung mit weiteren Informationen erkauft man sich zugegebenermaßen mit einem größeren

Speicherbedarf. Es existieren allerdings eine Reihe von Techniken, mit denen dieser gedrückt werden kann (für einen Überblick über die Implementierung von invertierten Listen siehe [WiMB99]).

8.3.3 Tries

Tries sind ebenfalls für Volltext-Retrieval konzipiert, dabei aber für Präfixsuche auf Zeichenketten optimiert worden.

Ein Trie (vom Wort Retrieval abgeleitet) ist eine baumartige Indexstruktur, in der Zeichenketten oder andere Schlüssel variabler Länge verwaltet werden. Ein Trie wird so gebaut, dass alle Zeichenketten, die das gleiche Präfix besitzen, durch denselben Knoten im Baum repräsentiert werden. Der Pfad von der Wurzel zum Knoten entspricht dabei diesem gemeinsamen Präfix. In den Nachfolgerknoten können alle existierenden Suffixe zu einem gegebenen Präfix gefunden werden. In Abb. 8-4 ist ein beispielhafter Trie zu sehen.

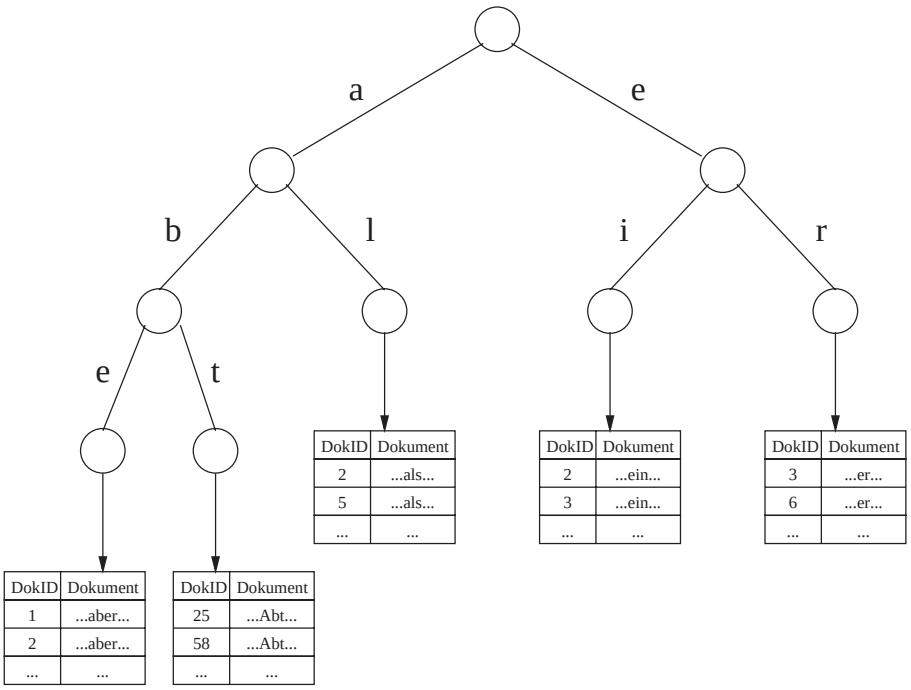


Abb. 8-4: *Trie*

Ein Vorteil von Tries ist die sehr kompakte Speicherung von Zeichenketten. Durch das Zusammenfassen der gemeinsamen Präfixe wird möglichst wenig an Daten redundant gehalten. Ein Nachteil ist allerdings, dass Tries eigentlich Hauptspeicherdatenstrukturen sind und dass die Umsetzung auf den Hintergrundspeicher mit Problemen behaftet ist (dazu mehr in 8.3.6).

Eine besondere Form von Tries sind Patricia-Bäume [Morr68]. Die Grundidee besteht darin, alle Schlüssel als Bitfolgen zu interpretieren. Ein Bit gibt dann an, ob an einem Knoten nach links oder nach rechts zu verzweigen ist. Außerdem wird in den inneren Knoten vermerkt, wie viele Bits beim Vergleich jeweils zu überspringen sind (falls alle gespeicherten Schlüssel in einem Zweig das gleiche Bitpräfix besitzen).

8.3.4 Kontextfilter

Die von Meuss und Strohmaier entwickelten Kontextfilter [MeSt99] stellen eine Erweiterung invertierter Listen dar. Jeder Eintrag wird zusätzlich mit Strukturinformationen versehen, die Aufschluss über den Pfad von der Wurzel zum jeweiligen Blattknoten geben. Aus Platzgründen werden diese Informationen stark komprimiert in einer Signatur abgelegt. Auf dem Volltextindex wird eine herkömmliche textuelle Suche durchgeführt. Anschließend wird in einem weiteren Schritt mit Hilfe der zusätzlichen Strukturinformation die zurückgelieferte Antwortmenge weiter reduziert.

Die Signatur zu einem Suchbegriff wird von Meuss und Strohmeier linearer Kontext genannt und speichert lediglich, welche Elemente auf dem Pfad liegen, nicht aber deren Reihenfolge. Pfadausdrücke in der Anfrage müssen nun jeweils in einen linearen Kontext übersetzt werden. Für alle Elemente, die im Anfragepfad vorkommen, werden im Anfragebitvektor die entsprechenden Bits gesetzt. Der Anfragebitvektor wird mit allen linearen Kontexten verglichen, die von der textuellen Anfragebearbeitung zurückgeliefert werden. Falls ein Element des Anfragepfads nicht in einem linearen Kontext eines Begriffs vorkommt, kann dieser verworfen werden. Da die Kodierung, wie bereits erwähnt wurde, verlustbehaftet ist, müssen die restlichen Antworten auf Korrektheit überprüft werden.

Buch	Autor	Titel	Text	Kapitel	Abschnitt
Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5

Abb. 8-5: *Beispiel für eine Bitmaske*

Zum Schluss illustrieren wir dieses Verfahren mit einem Beispiel. In Abb. 8-5 ist eine Bitmaske für lineare Kontexte im XML-Baum von Abb. 8-2 zu sehen. Der lineare Kontext des Begriffs »Einleitung« ist die Menge { Buch, Text, Kapitel, Titel }, kodiert als Bitvektor haben wir 101110.

Rein navigierende Anfragen werden nicht unterstützt, die Unterstützung kombinierter Anfragen ist jedoch möglich, wenn auf Inhalte zugegriffen wird. Dies wird durch herkömmliche Volltextindexe realisiert (wie invertierte Listen, angereichert mit zusätzlichen Informationen). Bei den Strukturbeschreibungen können Wurzelpfade und Platzhalter uneingeschränkt verwendet werden, andere Pfade werden unterstützt, sofern sie Teil eines Wurzelpfads sind. Da auf Signaturen aufgesetzt wird, kann anhand der Strukturbeschreibungen lediglich gefiltert werden.

8.3.5 SEQL/RCS

Auch diese beiden Ansätze unterstützen Volltext-Retrieval. Im Gegensatz zu Kontextfiltern ist auch die Auswertung von rein navigierenden Anfragen möglich, da auf die Tags unabhängig vom Inhalt zugegriffen werden kann. Wurzelpfadanfragen stellen kein Problem dar, Pfade mit Platzhaltern und von anderen Knoten aus werden unterstützt, sofern es darum geht, Vorgänger (ancestors) oder Nachfolger (descendants) von Knoten zu finden.

SEQL

Naughton et al. führen im Rahmen des Niagara-Projekts eine vereinfachte Anfragesprache ein: die Search Engine Query Language (SEQL) [Naug01]. Ziel ist es, XML-QL-Anfragen nach SEQL zu übersetzen und die SEQL-Anfragen auszuführen. Ein Vorteil von SEQL gegenüber XML-QL ist die schnellere Auswertung, da die Sprache wesentlich einfacher aufgebaut ist. Naughton et al. konzentrieren sich darauf, effiziente Indexstrukturen für SEQL zu entwickeln. Ähnlich wie lineare Kontexte übernimmt SEQL eine Filterfunktion, da die Antwort einer SEQL-Anfrage eine Obermenge der gesuchten Antwort darstellt. Die wesentlich komplexere XML-QL-Anfrage wird danach auf dem Ergebnis der SEQL-Anfrage ausgeführt. Das Ergebnis der SEQL-Anfrage ist dabei sehr viel kleiner als die ursprünglichen Daten. Wir beschreiben zunächst SEQL, danach die Übersetzung von XML-QL nach SEQL und schließlich eine Indexstruktur für die schnelle Auswertung von SEQL-Anfragen.

	SEQL-Anfrage	**Ergebnis (Alle XML-Dateien, die ...)**
Q1	»XML«	den Text »XML« enthalten.
Q2	Buch	ein Buch-Element enthalten.
Q3	Buch **contains** »Einf. in XML«	ein Buch-Element enthalten, das den Text »Einf. in XML« enthält.
Q4	Titel **is** »Einleitung«	ein Titel-Element mit dem Inhalt »Einleitung« enthalten.
Q5	Preis < 30.00	ein Preis-Element mit einem numerischen Inhalt kleiner als 30 enthalten.
Q6	**distance**(»Einleitung«, »Markup«) < 5	die Texte »Einleitung« und »Markup« in einem Abstand von weniger als fünf Worten enthalten.
Q7	»Einleitung« **and** »Markup«	den Text »Einleitung« und »Markup« enthalten.
Q8	Buch **contains** (»XML« **and** »Markup«)	ein Buch-Element mit den Texten «XML« und «Markup« enthalten.

Tab. 8-1: *SEQL-Anfragekonstrukte*

Eine atomare SEQL-Anfrage besteht lediglich aus einem (textuellen) Suchbegriff oder einem Element (siehe Anfrage Q1 und Q2 in Tab. 8-1). Atomare Anfragen können mit den binären Operatoren »contains«, »containedin« und »is« zu komplexeren Anfragen zusammengesetzt werden (siehe Q3 und Q4 in Tab. 8-1).

Weiterhin existieren Operatoren zum numerischen Vergleich und für Nachbarschaftsanfragen (siehe Q5 und Q6 in Tab. 8-1). Mit Hilfe von »and«, »or« und »except« können Antwortmengen vereinigt, geschnitten oder voneinander abgezogen werden (siehe Q7 und Q8 in Tab. 8-1). Mit dem speziellen Konstrukt »conformsto« kann geprüft werden, ob ein XML-Dokument den Richtlinien einer DTD entspricht.

Bei der Übersetzung von XML-QL werden aus Effizienzgründen nicht alle Nebenbedingungen der ursprünglichen Anfrage umgesetzt. Folgendes geht bei der Übersetzung verloren:

- XML-QL-Prädikate mit Negation werden ignoriert.
- Von den numerischen Vergleichsoperatoren werden nur =, <, ≤, >, ≥ übersetzt.
- Intervallausdrücke werden nicht berücksichtigt.
- Textvergleiche werden immer in »is«-Ausdrücke übersetzt (siehe Q4 in Tab. 8-1).

XML-QL-Prädikat	SEQL-Anfrage
A = 10 and not (B = »XML«)	A = 10
A = 10 or not (B = »XML«)	—
<Titel> Einleitung </Titel>	Titel **is** »Einleitung«
<Jahr> 1999 </Jahr>	Jahr = 1999

Tab. 8-2: *Übersetzung XML-QL nach SEQL*

Pfadausdruck	SEQL-Anfrage
A.B	A **contains** B
A.B*.C	A **contains** C
A.B+.C	A **contains** B **contains** C
A.(B|C)	(A **contains** B) or (A **contains** C)

Tab. 8-3: *Übersetzung XML-QL nach SEQL*

In Tab. 8-2 und Tab 8-3 ist das Übersetzungsschema sowohl von XML-QL-Prädikaten als auch von Pfadausdrücken angegeben.

Die SEQL-Indexstruktur umfasst drei invertierte Dateien (inverted files): eine Datei für Elemente, eine für Textsuchbegriffe und eine für DTDs. Für jedes Element wird eine Liste angelegt, in der vermerkt ist, in welchem XML-Dokument dieses Element auftritt und an welchen (Wort-)Positionen es beginnt und endet (siehe auch Abb. 8-6). Die Zugriffsstruktur für textuelle Daten ist ähnlich aufgebaut. Für jeden möglichen Suchbegriff existiert eine Liste, in der jeweils die ID des XML-Dokuments und die (Wort-)Position abgelegt sind, in denen der Suchbegriff auftritt (siehe ebenfalls Abb. 8-6). Die DTD-Listen sind noch einfacher aufgebaut. Hier wird lediglich für jede DTD eine Liste mit XML-Dokument-IDs gespeichert, die dieser DTD gehorchen.

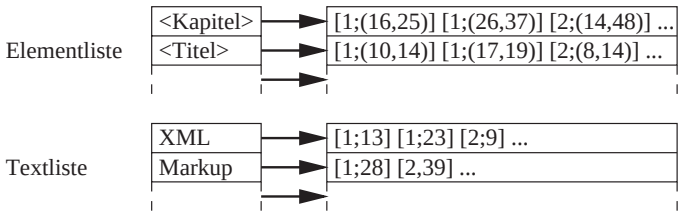


Abb. 8-6: *Invertierte Listen für SEQL*

Alle SEQL-Anfragen werden mit Hilfe der vorgestellten invertierten Listen bearbeitet. Jeder SEQL-Operator verlangt als Eingabe eine oder mehrere invertierte Listen und erzeugt als Ausgabe wiederum eine invertierte Liste. Die Anfragen Q1 und Q2 aus Tab. 8-1 liefern eine Suchbegriffs- bzw. Elementliste ungefiltert, also komplett, zurück. Bei den Anfragen Q3, Q4 und Q6 wird mit Hilfe der (Wort-) Positionen bestimmt, welche Listenelemente die Bedingung erfüllen. Diese werden dann zurückgeliefert, alle anderen ausgefiltert. Die Operatoren »and«, »or« und »except« werden jeweils durch Schnitt, Vereinigung und Differenz zweier invertierter Listen realisiert.

RCS

Die Idee, die im Niagara-System verfolgt wird, ist nicht neu. Bereits 1992 hat Burkowski so genannte Retrieval Command Strings (RCS) vorgestellt, die strukturelle Anfragen auf Dokumenten in Markup-Sprachen erlauben (SGML, ODA) [Burk92, Burk92(2)]. Es wird eine Algebra beschrieben, die ähnliche Operatoren wie SEQL enthält. Außerdem wird eine Vorverarbeitung auf invertierten Listen mit Filtern realisiert, d.h. auch hier existieren Listen mit den Positionen der Elemente und der Suchbegriffe in den Dokumenten.

8.3.6 Index Fabric

Im Gegensatz zu den bisherigen Ansätzen unterstützt ein Index Fabric kein Volltext-Retrieval, es ist ein Index für navigierende Anfragen bzw. kombinierte Anfragen, bei denen von Pfaden ausgegangen und zu Inhalten navigiert wird. Pfadausdrücke, die nicht von der Wurzel ausgehen, können mit Hilfe so genannter verfeinerter Pfade unterstützt werden. Die Auswertung von Pfadausdrücken mit Platzhaltern ist nicht effizient, da beim ersten Vorkommen eines Platzhalters auf eine umfassende Suche umgeschwenkt werden muss.

Cooper et al. kodieren Pfade in Zeichenketten und verwalten diese mit Hilfe einer neu entwickelten Indexstruktur, dem Index Fabric [CSFH01, CoSh01]. Dabei wird eine Variante eines Patricia Tries [Morr68] eingesetzt, um Zeichenketten kompakt und effizient zu speichern. Der Trie wird dabei so auf den Hintergrundspeicher abgebildet, dass eine dem B-Baum ähnliche, balancierte Struktur entsteht. In dieser Struktur werden Rohpfade (raw paths) und verfeinerte Pfade (refined paths) gespeichert. Rohpfade unterstützen die Auswertung von Ad-hoc-

Anfragen, während verfeinerte Pfade für spezielle bekannte Anfragetypen angelegt werden, um diese zu beschleunigen. Eine Anfrage wird (in beiden Fällen) in eine Zeichenkette umgewandelt, mit dem dann im Index Fabric gesucht wird. Wir werden zunächst den Aufbau der neuen Indexstruktur erklären und danach darauf eingehen, wie damit die Anfragebearbeitung von Pfadausdrücken beschleunigt werden kann.

Aufbau der Indexstruktur

In einem Index Fabric wird ein Patricia Trie beim Ablegen auf den Hintergrundspeicher in Blöcke eingeteilt, die von der Größe her den Seiten im Hintergrundspeicher entsprechen. Um bei einer Anfrageverarbeitung die relevanten Blöcke schnell zu finden, werden die Einstiegspunkte hierarchisch indexiert. Dafür werden ebenfalls Patricia Tries verwendet.

Solange der gesamte Patricia Trie in einen Block passt, unterscheidet sich ein Index Fabric nicht von einem regulären Patricia Trie. Bei einem Überlauf wird der Trie aufgespalten, und zwar so, dass als Ergebnis zwei in sich zusammenhängende Unterbäume entstehen. Aus diesem Grund wählt man eine Kante des Trie als Teilungskante (split edge), entlang der aufgeteilt wird. Außerdem sollte darauf geachtet werden, dass die beiden resultierenden Blöcke ungefähr gleich groß sind, um einen hohen Nutzungsgrad des Speichers zu erreichen. Die zwei entstandenen Blöcke werden auf einer höheren Stufe verwaltet. Der Trie auf der nächsthöheren Stufe wird konstruiert, indem der Teilungsknoten (split node), das ist der Elternknoten der Teilungskante, auf die nächsthöhere Stufe kopiert wird. Der Knoten bekommt einen Fernzeiger (far pointer) mit dem Bezeichner der Teilungskante und einen Direktzeiger (direct pointer). Der Fernzeiger zeigt auf den Block, in den die Teilungskante läuft, der Direktzeiger auf den Block, aus dem der Teilungsknoten kopiert wurde.

Bei der Aufspaltung des Blocks gibt es eine ganze Reihe von Spezialfällen zu beachten, auf die hier aber im Einzelnen nicht eingegangen werden soll (Details dazu sind in [CoSh01] zu finden). In Abb. 8-7 zeigen wir noch ein Beispiel für einen Index Fabric, um einen groben Eindruck davon zu vermitteln. Der Verständlichkeit wegen wurden die Kanten mit Buchstaben beschriftet, normalerweise wird in Patricia Tries mit der Binärdarstellung der Zeichenketten gearbeitet. Die Fernzeiger werden durch durchgezogene Linien, die Direktzeiger durch gepunktete Linien repräsentiert.

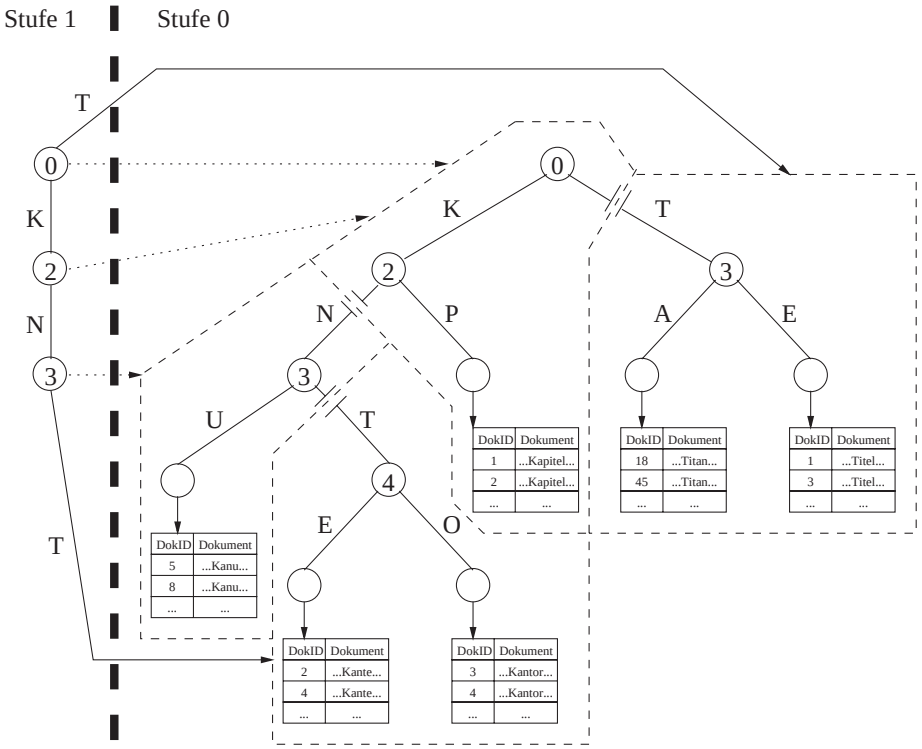


Abb. 8-7: *Beispiel für einen Index Fabric*

Eine Suche in einem Index Fabric beginnt am Wurzelknoten auf der höchsten Stufe. Innerhalb eines Blocks fährt die Suche in vom Patricia Trie gewohnter Weise fort, indem Buchstaben des Suchterms mit Kantenbezeichnern verglichen und die entsprechenden Kanten verfolgt werden. Bei Fernzeigern geht die Suche in einem Block auf einer tieferen Stufe weiter. Falls keine passende Kante gefunden wird, wird in dem Block weitergesucht, auf den der Direktzeiger verweist. Falls in Stufe 0 keine passende Kante mehr gefunden wird, bedeutet dies, dass dieser Suchterm in den Daten nicht existiert. Bei einem Datenzeiger muss auf die entsprechenden Daten zugegriffen werden, um sicherzustellen, dass die Suche erfolgreich war. Auf Grund der verlustbehafteten Komprimierung in Patricia Tries kann es zu falschen Treffern kommen.

Indexieren von Pfaden

Cooper et al. kodieren Pfade mit Hilfe von Designatoren, die aus speziellen Zeichen oder Zeichenketten bestehen. Jedem Element wird dabei ein eindeutiger Designator zugewiesen, z.B. β für <Buch>, α für <Autor> und τ für Titel. Die Zeichenkette βα`Erik T. Ray` hat die gleiche Bedeutung wie folgendes XML-Fragment:

```
<Buch>
   <Autor>
                Erik T. Ray
   </Autor>
</Buch>
```

Ein Designatorverzeichnis (designator dictionary) stellt eine Verbindung zwischen Designatoren und Elementen her.

Rohpfade sind Präfixe von Pfaden, die von der Wurzel zu einem Blatt laufen. Das XML-Fragment <A>Alpha<B>Beta<C>Gamma</C> </B> </A> z.B. wird durch drei Rohpfade repräsentiert: <A>Alpha, <A> <B>Beta und <A> <B> <C>Gamma. Vor der Abspeicherung in einem Index Fabric werden in den Rohpfaden die Elemente noch durch die entsprechenden Designatoren ersetzt.

Verfeinerte Pfade sind spezielle Pfade durch XML-Dokumente. Angenommen, es wird häufig nach den Autorennamen von Büchern gesucht, in denen Kapitel mit dem Titel *X* vorkommen. Dafür wird ein eigener Designator (beispielsweise ζ) eingeführt. Danach wird die relevante Information, die mittels Parsing aus dem XML-Dokument gewonnen wurde, mit Hilfe dieses Designators indexiert. Ein Eintrag für obiges Beispiel in einen Fabric-Index sähe so aus: ζ`Markup Konzepte Erik T. Ray`. Verfeinerte Pfade werden in Anfragen analog zu Rohpfaden verwendet.

Die Suche nach einem einfachen Pfadausdruck (der lediglich konstante Teilausdrücke enthält) erfordert eine Umwandlung des Pfadausdrucks in eine Zeichenkette unter Zuhilfenahme des Designatorverzeichnisses. Mit Hilfe dieser Zeichenkette wird nun eine Anfrage an einen Index Fabric gestartet. Etwas komplexere Anfragen können mit erhöhtem Aufwand ebenfalls bearbeitet werden. So wird $A.(B_1|B_2).C$ in zwei Anfragen $A.B_1.C$ und $A.B_2.C$ aufgeteilt und $A.*.C$ in eine Präfixanfrage auf A, gefolgt von einer umfassenden Suche.

8.4 Traditionelle Datenbanktechniken und Erweiterungen

Auch in diesem Abschnitt geben wir zuerst einige kurze Erläuterungen zu bereits bekannten Techniken, in diesem Fall B-Bäume (bzw. B^*-Bäume), und gehen danach auf Weiterentwicklungen ein, nämlich Extended Access Support Relations (und Varianten dazu) und multidimensionale Indexierung. Wir beschreiben jeweils kurz, welche Anfragetypen unterstützt werden.

8.4.1 B-Bäume / B^*-Bäume

B-Bäume [BaCr72] erlauben eine effiziente Unterstützung beim Zugriff auf einzelne Werte oder Bereichsanfragen. Einfache B-Bäume sind somit zunächst weder für Volltext-Retrieval noch für navigierende Anfragen besonders geeignet, es sei denn, es wird nur nach exakt festgelegten Werten gesucht. B-Bäume sind aber nicht so weit vom Bereich Information Retrieval entfernt, wie es auf den ersten

Blick den Anschein hat. So werden sie z.B. zur Verwaltung der Verzeichnisse von invertierten Dateien eingesetzt.

B-Bäume (bzw. die Weiterentwicklung B*-Bäume) sind die Standardindexstruktur relationaler Datenbanksysteme. Es handelt sich dabei um ausgeglichene Mehrwegbäume, d.h., im Gegensatz zu Binärbäumen kann ein Knoten mehr als einen Schlüssel und mehr als zwei Kinder haben. Die Schlüssel in einem Knoten *N* werden sortiert gehalten, und jedem Schlüssel wird ein Unterbaum zugeordnet. Alle Schlüssel in einem Unterbaum sind kleiner als die des zugeordneten Schlüssels und größer als die des Vorgängers des zugeordneten Schlüssels. In einem zusätzlichen Unterbaum werden alle Schlüssel gespeichert, die größer als die Schlüssel im Knoten *N* sind (siehe Abb. 8-8 für ein Beispiel).

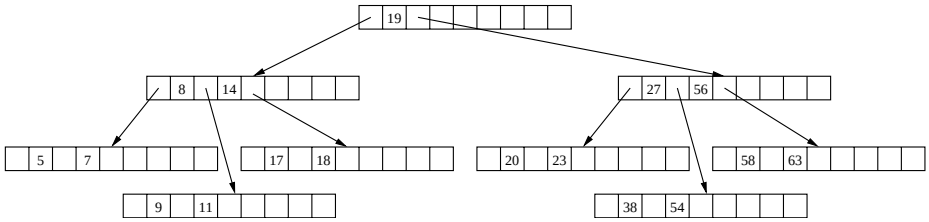


Abb. 8-8: *B-Baum*

In einem Datenbanksystem werden die Knoten eines B-Baums auf Seiten im Hintergrundspeicher abgebildet. Durch den größeren Verzweigungsgrad wird ein B-Baum wesentlich flacher als ein Binärbaum. Zusammen mit der Balancierung führt dies dazu, dass ein Schlüssel mit wenigen Seitenzugriffen gefunden werden kann. Um den Verzweigungsgrad weiter zu erhöhen, werden B*-Bäume eingesetzt. In B*-Bäumen werden alle Datensätze in den Blättern gehalten, in den inneren Knoten sind lediglich Referenzschlüssel zu finden. Da diese Schlüssel in der Regel wesentlich kleiner als Datensätze sind, wird auf diese Weise der Verzweigungsgrad erhöht und damit die Höhe des Baums verkleinert. Weitere Details zu B-Bäumen und B*-Bäumen sind z.B. in [HäRa01] und [KeEi97] zu finden.

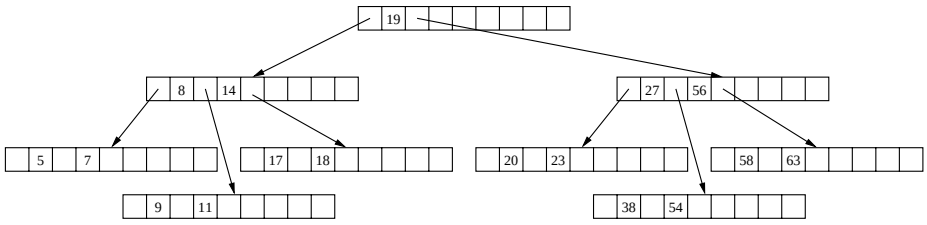
8.4.2 XASR/XISS

Eine erweiterte Zugriffsunterstützungsrelation (eXtended Access Support Relation oder kurz XASR [FiMo00]) und die Varianten davon besitzen einen herkömmlichen Index für Volltextanfragen und einen eigenen Index für den navigierenden Teil. Dieser Teil des Index ist in der Mächtigkeit der unterstützten Anfragen vergleichbar mit den SEQL/RCS-Ansätzen.

XASR

XASR ist ein Index, der die Eltern/Kind- und Vorgänger/Nachfolger-Beziehungen zwischen Knoten erhält. Im Unterschied zu den Ansätzen im vorherigen Kapitel

wird dabei aber nicht auf die textuelle Darstellung aufgesetzt (mit Abspeicherung der Positionen von Elementen und Suchbegriffen), sondern auf der Baumdarstellung. Ein XML-Dokumentbaum wird in Präorderreihenfolge traversiert, dabei werden die Knoten gekennzeichnet. Jedem Knoten wird dabei ein d_{min}-Wert zugewiesen, der angibt, wann der Knoten zum ersten Mal besucht wurde, und ein d_{max}-Wert, der angibt, wann der Knoten endgültig verlassen wurde. Dabei wird nicht zwischen Element- und Attributknoten unterschieden. Für jeden Knoten im Baum wird eine Zeile in einer XASR-Tabelle mit den Informationen zu d_{min}, d_{max}, dem Namen des Elements, der DokumentID und dem d_{min}-Wert des Elternknotens gehalten (siehe Abb. 8-9).



(a) XML-Dokumentbaum

d_{min}	d_{max}	eType	docID	parent_d_{min}
1	14	'Buch'	0	NULL
2	3	'Autor'	0	1
4	5	'Titel'	0	1
6	13	'Text'	0	1
7	12	'Kapitel'	0	6
8	9	'Titel'	0	7
10	11	'Abschnitt'	0	7

(b) XASR-Tabelle

Abb. 8-9: *XASR*

Die XASR wird mit einem Volltextindex (Full Text Index, kurz FTI) und einem Werteindex (Full Number Index, kurz FNI) kombiniert, um textuelle oder numerische Suchen auf den Knoteninhalten zu unterstützen. In diesen beiden Indexen werden die Vorkommen (docID und d_{min}) von Suchbegriffen und Zahlenwerten gespeichert. Bei einer Anfragebearbeitung werden die vom Volltext- bzw. Werteindex zurückgelieferten Knoten-IDs mit den entsprechenden Tupeln in der XASR verbunden. Dabei wird ein Pfadausdruck in einer Anfrage in eine Sequenz von Join-Operationen auf der XASR-Tabelle übersetzt. Diese Übersetzung läuft nach folgendem Schema ab. Je nach Navigationsachse ('/' oder '//') werden verschiedene Join-Prädikate angewandt (x_i und x_{i+1} sind dabei zwei Knoten auf dem Pfad):

- für '/':
  x_i.docID = x_{i+1}.docID ∧
  $x_i.d_{min}$ = x_{i+1}.parentID
- für '//':
  x_i.docID = x_{i+1}.docID ∧
  $x_i.d_{min} < x_{i+1}.d_{min}$ ∧
  $x_i.d_{max} > x_{i+1}.d_{max}$

Es können aber durchaus noch andere Anfragen unterstützt werden, wie z.B. solche, bei denen eine Variable an mehrere verschiedene Pfade gebunden wird.

XISS

Das XML-Indexierungs- und Speicherungssystem (XML Indexing and Storage System, kurz XISS) von Li und Moon [LiMo01] hat sehr große Ähnlichkeiten mit einer XASR. Es bietet folgende Erweiterungen:

- ein modifiziertes Nummerierungsschema
- Aufspaltung des Index in einen Element- und einen Attributindex

Das Nummerierungsschema ist so angepasst, dass spätere Einfügeoperationen in bestehende XML-Dokumente einfach auf den Index übertragen werden können. Jeder Knoten erhält einen Rang und eine Größe zugewiesen, so dass gilt (für ein Beispiel siehe Abbildung Abb. 8-10):

- Ist y ein Knoten und x der Elternknoten, so gilt Rang(x) < Rang(y) und Rang(y) + Größe(y) ≤ Rang(x) + Größe(x). Das Intervall von Rang(y) bis Rang(y) + Größe(y) ist also im Intervall von Rang(x) bis Rang(x) + Größe(x) enthalten.
- Für zwei Geschwisterknoten y und z (wobei y vor z bei einem Präorderdurchlauf besucht wird) gilt Rang(y) + Größe(y) < Rang(z).

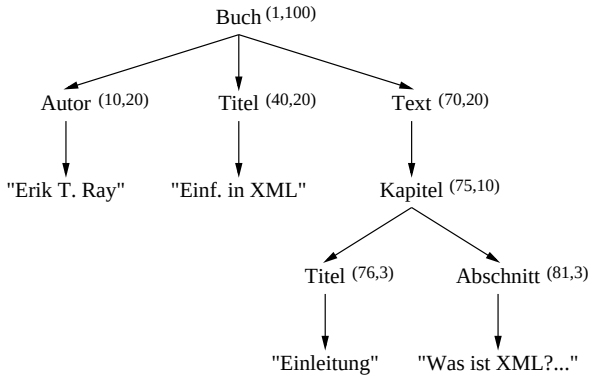


Abb. 8-10: *Nummerierungsschema für XISS*

Bei einer XASR wird zwischen Attributen und Elementen im Index nicht unterschieden. Das Nachsehen läuft in beiden Fällen über die gleiche Tabelle. Bei XISS werden dabei zwei verschiedene Tabellen verwendet.

Weitere Verbesserungen

path	pathID
/Buch	1
/Buch/Autor	2
/Buch/Titel	3
/Buch/Text	4
/Buch/Text/Kapitel	5
/Buch/Text/Kapitel/Titel	6
/Buch/Text/Kapitel/Abschnitt	7

Abb. 8-11: *Pfadrelation*

XASR und XISS benötigen einen Selbst-Join auf der Elementtabelle für jede Überprüfung der Enthaltenseinrelation, d.h. die Anzahl der Join-Operationen hängt von der Pfadlänge in der Anfrage ab. Seo, Lee und Kim schlagen folgende Idee vor [SeLK02]. Alle Pfade werden in einer Relation *Path(path, pathID)* gespeichert (siehe Abb. 8-11 für ein Beispiel), die Verbindung zu den Dokumenten wird über eine Relation *PathIndex(pathID, docID, begin, end)* realisiert. Volltextanfragen werden mit Relationen analog zu den FTI in XASR unterstützt.

Anfragen mit der Nachfolgerachse (descendant axis) '//' im Pfadausdruck werden mit Hilfe eines like-Prädikats in der where-Klausel formuliert. Dabei wird '//' durch '%/' ersetzt. Alle qualifizierenden Pfade werden danach mit nur einer einzigen Join-Operation mit der Relation *PathIndex* verbunden. Es folgen einige Beispiele für Übersetzungen von Pfadausdrücken:

- `/A/B` `Path.path = '/A/B'`
- `/A//B` `Path.path like '/A%/B'`
- `//A/B` `Path.path like '%/A/'B`
- `//A//B` `Path.path = '%/A%/B'`

8.4.3 Multidimensionale Indexierung

Auch die multidimensionale Indexierung ist nicht direkt für Volltext-Retrieval geeignet, da die Inhalte von Tags einfach in einem Attribut einer Relation abgelegt sind. Die Unterstützung navigierender Anfragen ist eingeschränkt auf XML-Dokumente mit DTDs, da zum Aufbau des Index die Pfade geordnet werden. Pfade, die ihren Ursprung nicht in der Wurzel haben, sind problematisch, ebenso Platzhalter (ab dem ersten Platzhalter wird die Anfrage in eine Bereichsanfrage aufgefächert).

Bayer verfolgt in seinem Ansatz, wie im Index Fabric, eine Kodierung von Pfaden, allerdings wird zur Indexierung der kodierten Pfade eine andere Methode vorgeschlagen [Baye01].

Zunächst werden Pfade, die mehrfach in einem Dokument vorkommen, durchnummeriert, um sie eindeutig zu identifizieren. (So wird der Pfad `Buch/Text/Kapitel/Titel` in Abb. 8-2 in die Bestandteile `Buch/Text/Kapitel[1]/Titel`, `Buch/Text/Kapitel[2]/Titel` usw. aufgespalten.) Jedes Dokument kann als Tupel mit beliebig vielen Attributen in einer universellen Relation XML-Rel interpretiert werden:

$$DokID\ AP_1\ AP_2\ AP_3\ \ldots\ AP_k,$$

wobei `Buch/Text/Kapitel[1]/Titel Einleitung` z.B. eine Ausprägung eines Attributpfads AP_i ist.

Da die explizite Abspeicherung aller Pfade einen hohen Speicherverbrauch mit sich bringen würde, werden zur Kodierung der einzelnen Pfade Surrogate eingeführt. Die Umsetzung von Pfadausdrücken in Surrogate geschieht mit Hilfe eines Verzeichnisses, aus dem ein beispielhafter Auszug in Tab. 8-4 zu sehen ist. Surrogate reflektieren dabei die von einer DTD vorgegebene Ordnung auf den Elementen.

Surrogat	Pfadausdruck
1.*	Autor
2	Titel
3	Text
3.*	Text/Kapitel
3.*.1	Text/Kapitel/Titel
3.*.2.*	Text/Kapitel/Abschnitt

Tab. 8-4: *Auszug aus einem Surrogatverzeichnis*

Da das relationale Modell keine Tupel mit variabler Attributanzahl vorsieht, wird die Relation XML-Rel auf eine Relation XML-Quad mit den Attributen DokID, Attributpfad, Attributwert und Surrogat abgebildet. Diese Relation XML-Quad wird wiederum in zwei Relationen XML-Ind und TypeDim aufgespalten. XML-Ind enthält dabei die Dokument-ID, das Surrogat und den Attributwert. In TypeDim ist der zu einem Surrogat gehörige Attributpfad zu finden und zusätzlich eine Typinformation zum Attributwert. In Tab. 8-5 ist ein Auszug aus einer XML-Ind-Relation zu finden.

DokID	Surrogat	Attributwert
1	1[1]	Erik T. Ray
1	2	Einführung in XML
1	3[1]1	Einleitung
1	3[1]2[1]	Was ist XML? …
…	…	…
1	3[2]1	Markup-Konzepte
1	3[2]2[1]	DieAnatomie …
1	3[2]2[2]	Elemente …
…	…	…

Tab. 8-5: *Auszug aus einer XML-Ind Relation*

Bayer stellt zusammenfassend fest, dass auf diese Weise eine kompakte Darstellung von XML-Dokumenten möglich ist. Dabei wird die Ordnung von Elementen innerhalb von Dokumenten in den Surrogaten widergespiegelt. Unterbäume in XML-Dokumenten entsprechen dabei Intervallen von Surrogaten.

Es wird vorgeschlagen, die Relation XML-Ind mit Hilfe einer mehrdimensionalen Indexstruktur (z.B. einem UB-Baum [RMFZ00]) zu indexieren, um die Anfrageauswertung zu beschleunigen. Dabei spannen die Attribute DokID, Surrogat und Attributwert einen dreidimensionalen Raum auf (siehe Abb. 8-12). Die Möglichkeit, auf Intervallen von Surrogaten suchen zu können, ist besonders sinnvoll, da auf diese Art und Weise gleichzeitig nach verschiedenen Instanzen des gleichen Pfads und in Unterbäumen gesucht werden kann.

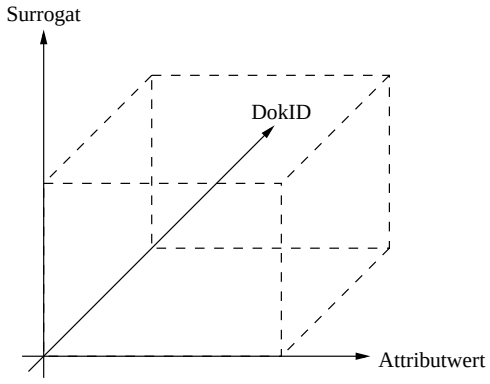


Abb. 8-12: *XML-Ind im dreidimensionalen Raum*

8.5 Originäre Indexstrukturen

In diesem Abschnitt werden schließlich neue Ansätze vorgestellt, die nicht direkt einem der bisherigen Bereiche zugeordnet werden können. Es handelt sich dabei um Data Guides, T-Indexe und Tree-Matching.

8.5.1 Data Guides

Das Datenbanksystem Lore [HAGQ97, HWAL98], für das die Data Guides entwickelt wurden, enthält ein ganzes Konglomerat aus verschiedenen Indexstrukturen, darunter auch ein Index für Volltext-Retrieval. Data Guides sind für die effiziente Bearbeitung von navigierenden und kombinierten Zugriffen zuständig. Dabei stellen Wurzelpfade keine Hindernisse dar, während die Abwicklung von Anfragen mit Nichtwurzelpfaden und Platzhaltern problematischer ist, da das Ganze dann auf eine umfassende Suche hinausläuft.

Lore setzt auf dem Object Exchange Model (OEM) auf, einem Modell, das nicht nur für XML entworfen wurde, sondern allgemein für semistrukturierte Daten. In OEM sind die Knoten Objekte mit einer eindeutigen Kennung (identifier). Es wird unterschieden zwischen atomaren Objekten, die keine ausgehenden Kanten besitzen, und komplexen Objekten, die beliebig viele ausgehende Kanten besitzen. Bezeichner (labels) markieren die Kanten zwischen den Knoten. XML-Dokumente können auf OEM-Graphen abgebildet werden, indem die XML-Elemente nicht auf Knoten, sondern auf Kanten projiziert werden. Abb. 8-13 zeigt die Darstellung des XML-Dokuments aus Abb. 8-1 als OEM-Graph (der Einfachheit halber wurden die Kennungen der Knoten weggelassen).

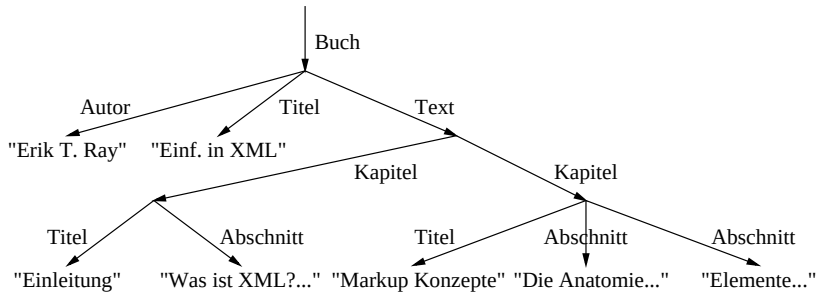


Abb. 8-13: *Beispiel für die Darstellung als OEM-Graph*

Lorel, die Anfragesprache für das Lore-Datenbanksystem, führt außerdem eine automatische Typumwandlung (type coercion) durch. Um Anfragen für verschiedene Datentypen effizient zu unterstützen, existieren in Lore vier verschiedene Indexe: ein Index für Werte (Value Index, kurz VIndex), ein Index für Text (Text Index, kurz TIndex), ein Index zum Finden von Elternknoten (Link Index, kurz LIndex) und ein allgemeinerer Pfadindex (Path Index, kurz PIndex).

Ein VIndex indexiert Knoten mit einem Inhalt vom Typ Ganzzahl, Fließkommazahl oder Zeichenkette, die durch eine Kante mit Bezeichner *l* erreichbar sind. Die Kantenbezeichner werden mit in den Index aufgenommen. Zusätzlich dazu werden die Daten auch nach ihnen partitioniert, da man davon ausgeht, dass zum Zugriffszeitpunkt der Bezeichner bekannt ist. Beim Einrichten eines Index für Bezeichner *l* werden drei VIndexe aufgebaut: ein Index für Fließkommazahlen (und für Ganzzahlen, die in Fließkommazahlen umgewandelt wurden), einer für Zeichenketten und einer für nach Fließkommazahlen umgewandelte Zeichenketten. Jeder dieser drei Indexe wird mit Hilfe eines B^*-Baums realisiert.

Für Textsuche reichen einfache Vergleichsoperatoren wie =, <, > oft nicht aus. Deswegen werden aufwändigere Textsuchen mit Methoden unterstützt, die im Bereich Information Retrieval üblich sind. Bei Lore verbirgt sich hinter dem Textindex (kurz TIndex) eine invertierte Liste, die sowohl den Knoten (mit dem jeweiligen Kantenbezeichner) als auch die Position jedes indexierten Textstücks speichert.

Ein Linkindex (LIndex) stellt einen Mechanismus zur Verfügung, um alle Elternknoten eines Knotens zu finden. Für einen Kindknoten *c* und einen Kantenbezeichner *l* gibt ein LIndex alle Elternknoten *p* zurück, so dass eine Kante *l* zwischen *p* und *c* existiert. Realisiert wird ein LIndex durch eine erweiterbare Hash-Tabelle (extendable hashing).

Der interessanteste Index in Lore ist der Pfadindex (Path Index, kurz PIndex). Mit Hilfe dieses Index können alle Knoten ausgemacht werden, die auf einem bestimmten Pfad erreichbar sind. Dabei wird eine Zusammenfassung aller momentan möglichen Pfade in einem so genannten Data Guide gehalten. Data Guides decken die zugrunde liegende Organisation der Daten auf und beschreiben sie in einer kompakten Weise. Bevor wir zur Definition von Data Guides kommen, führen wir noch einige einleitende Definitionen ein.

Definition 8.5.1 *Ein Bezeichnerpfad (label path) eines Knotens o_i ist eine Sequenz von durch Punkte getrennten Bezeichnern $l_1.l_2....l_n$, so dass es ausgehend von o_i einen Pfad im Graph entlang der Kanten e_1 bis e_n (mit den Bezeichnern l_1 bis l_n) gibt.*

Definition 8.5.2 *Ein Datenpfad (data path) eines Knotens ist eine alternierende Sequenz von Kantenbezeichnern und Knotenkennungen in der Form $\xrightarrow{l_1} o_1 \xrightarrow{l_2} o_2 ... \xrightarrow{l_n} o_n$, so dass es einen Pfad im Graph entlang der Kanten e_1 bis e_n mit den Bezeichnern l_1 bis l_n gibt, bei dem die Knoten x_1 bis x_n mit den Kennungen o_1 bis o_n durchschritten werden.*

Definition 8.5.3 *Ein Datenpfad d ist eine Instanz eines Bezeichnerpfads l, wenn beide die gleiche Sequenz der Bezeichner l_1 bis l_n besitzen.*

Definition 8.5.4 *Eine Zielmenge (Target Set) $T_s(l)$ ist die Menge aller Knoten eines OEM-Graphs s, die über den Pfad l erreichbar sind.*

Nach diesen einleitenden Definitionen kommen wir nun zu den Data Guides selbst [GoWi97].

Definition 8.5.5 *Ein Data Guide für einen Graph s ist ein Graph d, so dass jeder Bezeichnerpfad in s eine Datenpfadinstanz in d hat und jeder Bezeichnerpfad in d auch ein Bezeichnerpfad in s ist.*

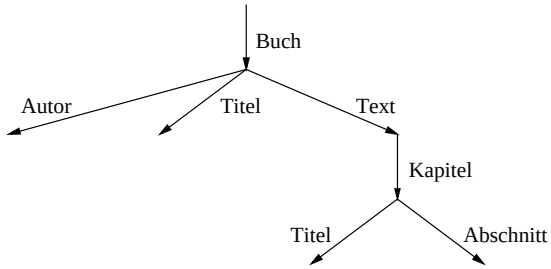


Abb. 8-14: *Beispiel für ein Data Guide*

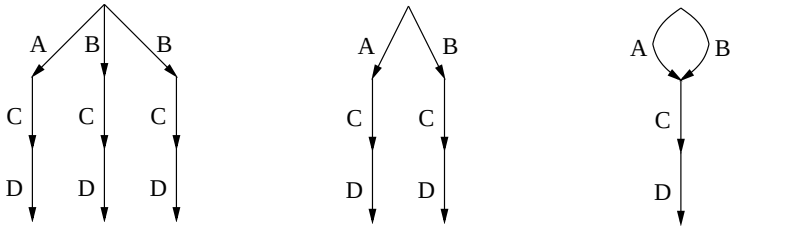
Ein OEM-Graph eines semistrukturierten Dokuments kann als nicht deterministischer endlicher Automat (NEA) interpretiert werden. Dabei entsprechen die Knoten den Zuständen des Automaten und die Kanten den Zustandsübergängen. Das Aufstellen eines Data Guide ist quasi die Umwandlung dieses NEA in einen deterministischen endlichen Automat (DEA). Wenn der Ausgangsgraph ein Baum ist, kann dies in linearer Zeit geschehen. Im schlimmsten Fall hat der Algorithmus allerdings exponentiellen Aufwand. Es kann durchaus mehrere DEAs für einen NEA geben. Auf den ersten Blick mag es verlockend erscheinen, den Algorithmus für die Umwandlung in einen minimalen DEA zu verwenden. Dies sollte aber nicht unbedingt angestrebt werden, da dann das spätere Ändern des Data Guide mit erheblichem Aufwand verbunden sein kann.

Eine Aufgabe von Data Guides ist es, die Struktur einer Sammlung von XML-Dokumenten zusammenzufassen. Durch Abspeicherung von zusätzlichen Informationen kann die Funktionalität noch erweitert werden. So ist es interessant, direkten Zugriff auf Knoten zu bekommen, die auf einem Pfad *l* erreichbar (d.h. die in einer Zielmenge enthalten) sind. Ein Data Guide garantiert, dass jeder Bezeichnerpfad im ursprünglichen Graph genau einen Knoten im Data Guide erreicht. Dabei kann es aber vorkommen, dass mehrere (verschiedene) Bezeichnerpfade den gleichen Knoten im Data Guide erreichen, selbst wenn sie im Originalgraphen verschiedene Zielmengen haben.

Aus diesem Grund werden starke Data Guides (strong data guides) eingeführt. Man ist an Data Guides interessiert, in denen jede Menge von Bezeichnerpfaden, die auf dieselbe Zielmenge im Data Guide verweisen, dies auch im Originalgraphen tun.

Definition 8.5.6 *Sei s ein OEM-Graph und d ein zugehöriger Data Guide. Weiterhin sei gegeben ein Bezeichnerpfad l in s, sei $T_s(l)$ die Zielmenge von l in s und $T_d(l)$ die Zielmenge von l in d. Sei $L_s(l)=\{m \mid T_s(m)=T_s(l)\}$, d.h. $L_s(l)$ ist die Menge aller Bezeichnerpfade in s, die die gleiche Zielmenge wie l haben. Analog dazu sei*

$L_d(l)=\{m \mid T_d(m)=T_d(l)\}$. Wenn für alle Bezeichnerpfade l in s $L_s(l)=L_d(l)$, dann ist d ein starker Data Guide.



(a) Originaldokument (b) nichtstarker Data Guide (c) starker Data Guide

Abb. 8-15: *Unterschied zwischen starken und nicht starken Data Guides*

Anders formuliert ist ein starker Data Guide die Transformation des ursprünglichen Graphen, interpretiert als NEA, in einen deterministischen Potenzmengenautomaten.

Data Guides werden für verschiedene Zwecke in Lore eingesetzt. Ein wichtiger Punkt ist die Schemaerkennung. Um überhaupt sinnvolle Anfragen an eine XML-Datenbank stellen zu können, muss man erst einmal etwas über die Organisation der Daten in Erfahrung bringen. Mit Hilfe eines Data Guide kann man alle derzeit möglichen Pfade aller gespeicherten Dokumente anzeigen und durchlaufen, ohne die gesamte Datenbank durchsuchen zu müssen. In Lore wurde eine Schnittstelle entwickelt, mit der ein Data Guide graphisch zum Browsen dargestellt wird.

Ein weiteres Gebiet ist die Unterstützung von Pfadanfragen. Bei Anfragen auf XML-Daten kommen sehr häufig Prädikate vor, die sich auf Pfade in XML-Dokumenten beziehen bzw. auf Bedingungen, die die Pfade erfüllen sollen. Im Moment unterstützen Data Guides Wildcards in Pfadausdrücken noch nicht direkt, aber es ist immer noch günstiger, ein Data Guide komplett zu durchlaufen, als die gesamte Datenbank.

Zu guter Letzt können Data Guides auch zur Anfrageoptimierung eingesetzt werden. So kann man schnell prüfen, ob eine Anfrage oder Unteranfrage ein leeres Ergebnis zurückliefert, da der in der Anfrage angegebene Pfad überhaupt nicht existiert.

8.5.2 T-Index

Der von Milo und Suciu entwickelte Schablonenindex (Template Index, kurz T-Index) stellt eine Erweiterung von Data Guides dar [MiSu99], d.h., für Volltext-Retrieval ist ein eigener Index nötig. Für navigierende und kombinierte Anfragen stellen Wurzelpfade weiterhin kein Problem dar, und im allgemeinsten Fall des T-Index werden auch Nichtwurzelpfade und Platzhalter unterstützt.

Wir führen den T-Index in mehreren (motivierenden) Schritten ein. Zunächst wird ein so genannter 1-Index angegeben, eine einfache Abwandlung der Data

Guides. Dieser wird im nächsten Schritt zu einem 2-Index erweitert, um schließlich zu einem T-Index verallgemeinert zu werden.

1-Index

Da es sich bei den starken Data Guides aus dem vorigen Abschnitt um deterministische Potenzmengenautomaten handelt, kann die Größe eines Data Guide exponentiell sein bzw. der Aufbau exponentiellen Aufwand verursachen. Milo und Suciu lockern die Beschränkungen in Bezug auf Determinismus, indem sie den ursprünglichen Datengraph *s* nicht vollständig in einen deterministischen Automaten umwandeln. Auf diese Weise wird der Index kompakter als ein Data Guide.

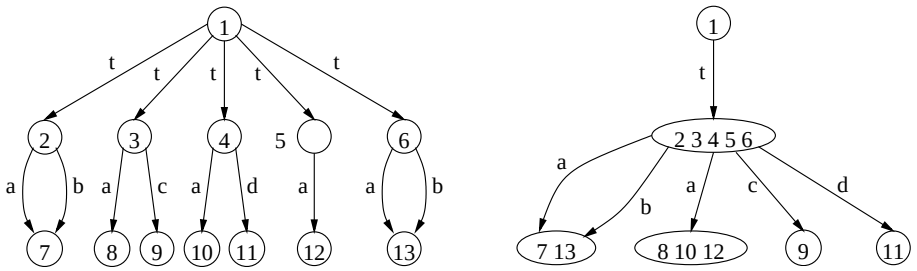
Milo und Suciu teilen dafür die Knoten des Ursprungsgraphen in Äquivalenzklassen ein. Für zwei Knoten v und u gilt:

$$v \equiv u \Leftrightarrow L_v = L_u$$

mit

$$L_v(DB) := \{w \mid w = l_1 ... l_n, \exists \text{Pfad } v_0 \xrightarrow{l_1} ... \xrightarrow{l_n} v \text{ mit Wurzel } v_0\}$$

Zwei Knoten gehören zur selben Äquivalenzklasse, wenn sie über die gleiche Menge von Pfaden erreichbar sind. Abb. 8-16 zeigt einen Graphen mit zugehörigem 1-Index. Der entsprechende starke Data Guide ist in Abb. 8-17 zu sehen.



(a) Originaldokument (b) 1-Index

Abb. 8-16: *Äquivalenzklassen im 1-Index*

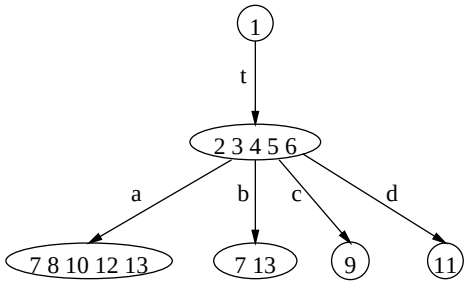


Abb. 8-17: *Starker Data Guide*

Da die Berechnung von Äquivalenzklassen auf Graphen sehr teuer ist [StMe73], schlagen Milo und Suciu vor, Verfeinerungen der Äquivalenzrelation ≡ zu verwenden. Eine Äquivalenzrelation ≈ ist eine Verfeinerung einer anderen Äquivalenzrelation, wenn $v \approx u \Rightarrow v \equiv u$. Dabei werden zwei Verfeinerungen in Betracht gezogen: Bisimulation ($\approx_b$) und Simulation ($\approx_s$). Ziel ist es, (Sprach-)Äquivalenzklassen abzuschätzen.

1-Indexe bieten die gleiche Funktionalität wie Data Guides. Sie haben den Vorteil, dass sie im schlimmsten Fall linear in der Größe des Originalgraphen sind (Data Guides sind im schlimmsten Fall exponentiell). Der Nachteil ist, dass beim Durchlaufen der Pfade während einer Anfragebearbeitung eventuell mehr als ein Zweig des 1-Index verfolgt werden muss. Die gefundenen Knotenmengen der gesuchten Äquivalenzklassen müssen danach noch vereinigt werden. Falls es sich beim Ursprungsgraphen um einen Baum handelt, sind 1-Indexe mit Data Guides identisch.

2-Index

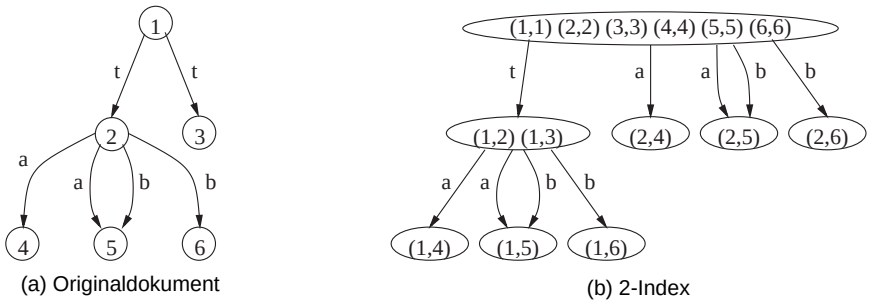
Mit Hilfe eines 1-Index können Anfragen mit Ausdrücken der Form $P\ x$ unterstützt werden, wobei es sich bei P um einen Pfadausdruck und bei x um einen Knoten handelt. Wenn man aber an der Bearbeitung von Ausdrücken der Form $*\ x_1\ P\ x_2$ interessiert ist, hilft ein 1-Index nicht weiter. In diesem Fall benötigt man eine Äquivalenzrelation zwischen Paaren von Knoten, da zwei Variablen gebunden werden müssen, also

$$(v, u) \equiv (v', u') \Leftrightarrow L_{(v,u)} = L_{(v',u')}$$

mit

$$L_{(v,u)}(DB) = \{w \mid w = l_1 \ldots l_n, \exists \text{Pfad } v \xrightarrow{l_1} \ldots \xrightarrow{l_n} u \text{ in } s\}$$

Ein 2-Index beschleunigt die Auswertung von Anfragen in obiger Form. Er sieht folgendermaßen aus: Die Wurzeln sind alle Äquivalenzklassen der Form $[(x, x)]$. Für jede Kante l zwischen u und u' und jedem Knoten v in DB existiert eine Kante l zwischen $[(v, u)]$ und $[(v, u')]$. In Abb. 8-18 ist ein Graph mit zugehörigem 2-Index zu sehen.



(a) Originaldokument

(b) 2-Index

Abb. 8-18: *Datengraph mit 2-Index*

Auch hier schlagen Milo und Suciu aus Effizienzgründen eine Verfeinerung $\approx$ vor, für die gilt

$$(v, u) \approx (v', u') \Rightarrow (v, u) \equiv (v', u')$$

T-Index

Ziel eines T-Index ist es, die Bearbeitung von Anfragen mit Pfadausdrücken der Form $P_1\,x_1\,P_2\,x_2 \ldots P_n\,x_n$ effizient zu unterstützen. Auch hier werden wieder Äquivalenzklassen zwischen Tupeln von Knoten aufgebaut. Wir möchten, dass zwei Tupel der Form $(u_1,u_2,\ldots,u_n)$ und $(v_1,v_2,\ldots,v_n)$ in verschiedenen Äquivalenzklassen liegen, wenn sie auf verschiedenen Anfragepfaden $P_1\,x_1\,P_2\,x_2 \ldots P_n\,x_n$ erreicht werden.

Aus Platzgründen kann hier nicht das gesamte, sehr komplexe Verfahren zum Aufbau eines T-Index beschrieben werden. Die wichtigsten Ideen und Ergebnisse werden wir kurz anreißen (für Details siehe [MiSu99]). Im Unterschied zu einem 1-Index oder 2-Index unterstützt ein T-Index Anfragen mit mehr als einem Pfadausdruck *P*. Bei einem T-Index müssen die Variablen x_1, x_2,…, x_n gebunden und die verschiedenen Teile P_1 bis P_n voneinander abgegrenzt werden. Die Äquivalenzklassen werden nach und nach über partielle Bindungen der Variablen gebildet. Der Abschluss einer Abarbeitung eines Teilpfadausdrucks P_i wird mit einem speziellen Trennsymbol, \$, gekennzeichnet. Im Graph eines T-Index existieren entsprechende \$-Kanten, die diesen Übergang repräsentieren. Durch Verfolgen einer \$-Kante geht man zum nächsten Teilpfadausdruck über. Im Gegensatz zu 1- und 2-Indexen wird beim T-Index auch zwischen Terminal- und Nichtterminalknoten unterschieden, wobei nur Knoten mit vollständiger Variablenbindung Terminalknoten sein können. Gemeinsam mit 1- und 2-Indexen ist die Verwendung von Verfeinerungen von Äquivalenzrelationen aus Effizienzgründen.

Wie werden T-Indexe zur Anfragebearbeitung eingesetzt? Das einfachste Szenario liegt vor, wenn der Pfadausdruck der Anfrage auf die Schablone des T-Index passt. Wir haben in diesem Fall eine Anfrage mit einem Pfadausdruck der Form $q = P_1\,x_1\,P_2\,x_2 \ldots P_n\,x_n$, d.h., es werden *n* Variablen gebunden. Die Anfrage auf dem T-Index wird mit dem Ausdruck $P_q = P_1.\$.P_2.\$.\ldots.P_n$ gestartet und es wird geprüft, ob das Wort P_q vom T-Index-Automaten akzeptiert wird. Da P_q genau *n*-1 \$-Zeichen enthält, werden im Akzeptanzfall Terminalknoten erreicht. Die Antwort auf die Anfrage ist die Vereinigung aller *n*-Tupel in den erreichten Knoten.

Einerseits stellt ein T-Index somit eine Erweiterung des 1- und 2-Index auf Ausdrücke dar, in denen *n* Anfragevariablen gebunden werden. Andererseits kann dies auch als Spezialisierung gesehen werden, da Anfragen nur dann direkt mit Hilfe des Index verarbeitet werden können, wenn ihre Pfadausdrücke zur Schablone passen. Milo und Suciu zeigen jedoch, dass T-Indexe eventuell auch zur Bearbeitung von Anfragen eingesetzt werden können, die nicht zur Schablone passen.

Das generelle Problem, zu entscheiden, ob eine Pfadanfrage *q* passend auf T-Indexe umgeschrieben werden kann, verallgemeinert das Query-Rewriting-Problem auf reguläre Pfadausdrücke. Da dies ein offenes Problem ist, beschränken

sich Milo und Suciu auf eine Präfixersetzung. Dabei wird der Pfadausdruck der Anfrage so zerlegt, dass ein Präfix $P_1\,x_1\,P_2\,x_2 \ldots P_i\,x_i$ mit Hilfe von bestehenden T-Indexen auswertbar ist. Milo und Suciu zeigen, dass das Problem, zu entscheiden, ob eine Präfixersetzung existiert, PSPACE-vollständig ist. Für Spezialfälle von Anfragepfaden und T-Indexschablonen mit einfachen Wildcards können Ersetzungen in PTIME gefunden werden.

Ein weiterer Nachteil ist der Aufwand, der zum Aufbau eines T-Index nötig ist. Deswegen schlagen Milo und Suciu vor, T-Indexe für spezielle Anfragetypen mit konstanten Ausdrücken zu konstruieren. Dies ergibt Sinn, wenn z.B. bekannt ist, dass häufig Anfragen mit Prädikaten folgender Form gestellt werden: `*.Restaurant.`x_1`.P.`x_2 oder `*.(Unterabteilung*).`x_1`.P.`x_2.

8.5.3 Tree-Matching

Volltext-Retrieval muss auch bei diesem Ansatz über einen eigenen (zusätzlichen) Index realisiert werden. Es ist nicht ganz richtig, hier von navigierenden und kombinierten Zugriffen zu sprechen, da es sich um eine andere Art der Anfrageformulierung handelt. Wenn man Pfade als degenerierte Bäume ohne Verzweigung interpretiert, lassen sich folgende Aussagen machen: Rein navigierende Anfragen werden von diesem Ansatz noch nicht unterstützt, die Baummuster in einer Anfrage müssen mit Inhaltsangaben zu Knoten versehen werden. Auch Platzhalter werden nicht berücksichtigt. Dafür ist es möglich, Baumstrukturen in einer Anfrage anzugeben.

Es gibt Ansätze, die darüber hinausgehen, nur Pfadausdrücke in Anfragen zuzulassen. Einer dieser Ansätze ist das Suchen nach Bauminklusionen. Die zentrale Idee dabei ist, dass der Benutzer sein Wissen über die Struktur der Anfrage in Form eines Baummusters angeben kann. Dieses Baummuster wird mit semistrukturierten Dokumenten verglichen, und alle (Teil-)Dokumente, die das Baummuster enthalten, werden als Antwort zurückgeliefert. Kilpeläinen untersuchte zehn verschiedene Varianten von Bauminklusionen in [Kilp92], deren Berechnung aber zumeist eine hohe Komplexität aufweist. Als Mindestanforderung für die Benutzung als Grundelemente in Anfragen verlangen Kilpeläinen und Mannila in [KiMa93] die Erhaltung der Bezeichner (labels) und der Vorgänger-/Nachfolgerbeziehungen. Als Kompromiss zwischen Mächtigkeit und Komplexität schlagen sie deshalb eine geordnete Bauminklusion (ordered tree inclusion, kurz OTI) und eine ungeordnete Bauminklusion (unordered tree inclusion, kurz UTI) vor. Bei OTI muss die horizontale Ordnung der Elemente zusätzlich erhalten bleiben. Abb. 8-19 zeigt eine OTI und eine UTI eines Anfragemusters.

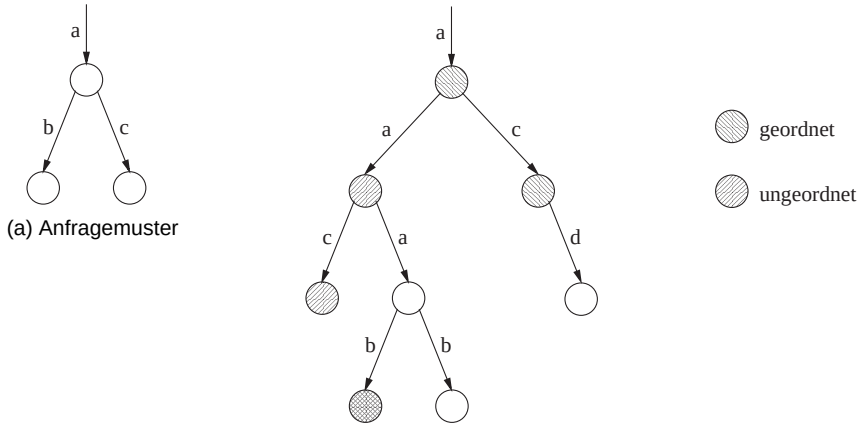


Abb. 8-19: *Anfragemuster mit OTI und UTI*

Ein weiteres Problem bei der Anfragebearbeitung von Bauminklusionen stellt die potenzielle Größe des Ergebnisses dar, d.h., allein das Aufbauen jeder einzelnen Antwort kann exponentiellen Aufwand haben. Meuss führt in [Meus98] CARs (Complete Answer Representations) ein, die eine kompakte Darstellung erlauben, und erläutert außerdem, wie die Bearbeitung von Bauminklusionen mit Hilfe von Indexstrukturen beschleunigt werden kann.

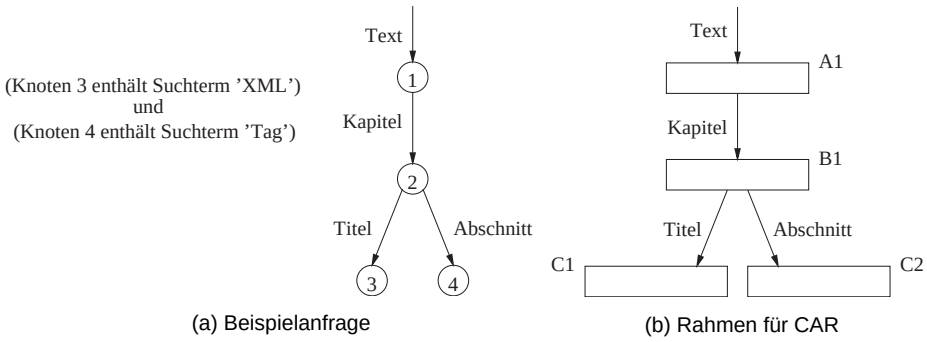


Abb. 8-20: *Beispielanfrage mit Rahmen*

Wir führen die Funktionsweise der Anfragebearbeitung mit Hilfe eines Beispiels vor. In Abb. 8-20 ist auf der linken Seite eine Beispielanfrage zu sehen. Die Anfrage besteht nicht nur aus dem Baummuster, sondern noch aus zwei Prädikaten, die die Antwort weiter einschränken. Auf der rechten Seite der Abbildung ist der Rahmen (Framework) zu sehen, in dem die CAR aufgebaut wird. In einem Rahmen entsprechen die Kästen (Slots), Kanten und Bezeichner den Knoten, Kanten und Bezeichnern des Anfragemusters.

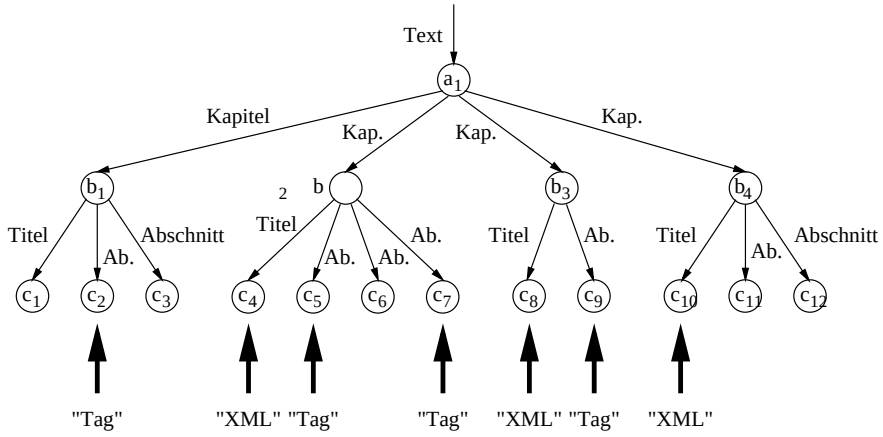


Abb. 8-21: *Beispieldaten*

Über einen kombinierten Volltext-/Pfadindex (wie dem multidimensionalen Index aus Abschnitt 8.4.3) werden zuerst alle Knoten, die den Suchterm XML enthalten, zusammen mit ihrem Pfad geholt. Falls der geholte Pfad mit dem Pfad `Text/Kapitel/Titel` übereinstimmt (Wildcards werden in [Meus98] nicht berücksichtigt), dann wird dieser zusammen mit den entsprechenden Knoten in den Rahmen eingetragen. Angenommen, die Anfrage wird auf den Daten in Abb. 8-21 ausgeführt (die mit »XML« und »Tag« markierten Kanten sind die Einstiegspunkte über den Index). Nach dem ersten Schritt sieht der Rahmen wie in Abb. 8-22 dargestellt aus.

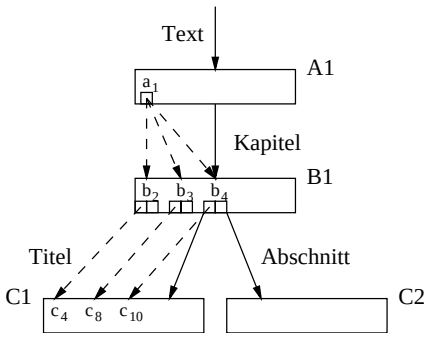


Abb. 8-22: *Rahmen nach erstem Schritt*

In einem zweiten Schritt werden, analog zum ersten, alle Knoten mit ihren Pfaden geholt, die den Suchterm »Tag« enthalten. Im zweiten (und allen weiteren) Schritten muss das Ergebnis nach Aussortieren der irrelevanten Pfade mit dem Teilergebnis verbunden werden, das bereits in den Rahmen eingetragen ist. Bei der logischen Und-Verknüpfung in der Beispielanfrage bedeutet dies, dass die Knotenmengen miteinander geschnitten werden. Abb. 8-23 zeigt den Rahmen nach Auswerten des zweiten (und in diesem Beispiel letzten) Schrittes. Aus diesem Rahmen können die einzelnen konkreten Antwortbäume generiert werden.

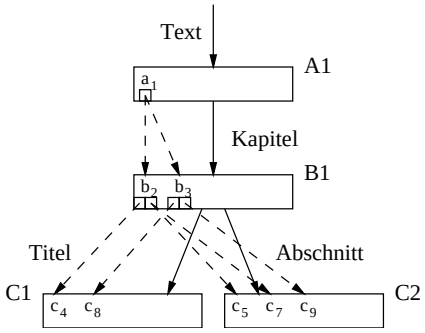


Abb. 8-23: *Endgültiger Rahmen*

8.6 Kurze Evaluation

Die wichtigsten Ergebnisse des Vergleichs sind in Tab. 8-6 zusammengefasst. Da sie nicht alle ohne weiteres auf eine einfache Bewertung reduzierbar sind, wurden sie an einigen Stellen kommentiert.

			Navig. Anfrage			Komb. Anfrage					
	WA	**VT**	**WP**	**NP**	**PH**	**WP**	**NP**	**PH**	**Ex**	**Eff**[a)]	**UD**
Inv. Dateien	o[b)]	+	–	–	–	–	–	–	+	o	o
Kontextfilter	o[b)]	+	–	–	–	+	o	+	–	o[c)]	o[c)]
SEQL/RCS	o[b)]	+	+	o	+	+	o	+	–	o[c)]	o[c)]
Index Fabric	o[b)]	o[d)]	+	+	–	+	+	–	–	o	+
B*-Bäume	+	–	–	–	–	–	–	–	+	+	+
XASR/XISS	+	+	+	+	+	+	+	+	+	o	+
Multidim.	+	o[d)]	+	–	–	+	–	–	+	+[e)]	o[f)]
Data Guides	o[b)]	o[d)]	+	–	–	+	–	–	+	o	o[g)]
T-Index	o[b)]	o[d)]	+	+	+	+	+	+	+	–	–
Tree-Match	o[b)]	o[d)]	–	–	–	+	+	–	+	o[h)]	o[h)]

Legende: WA=Werteanfrage, VT=Volltext-Retrieval, WP=Wurzelpfadanfragen, NP=Nichtwurzelpfadanfragen, PH=Platzhalter, Ex=Exaktheit, Eff=Effizienz, UD=Update-Fähigkeit

[a)] Ein direkter Vergleich (sowohl analytisch als auch experimentell) steht noch aus.

[b)] Exakte Werte zu finden ist möglich, Bereichsanfragen jedoch nicht.

[c)] Hängt von der jeweiligen Implementierung der invertierten Liste ab.

[d)] Das Hinzufügen eines Volltextindex sollte kein großes Problem darstellen.

[e)] Angenommen, eine effiziente Implementierung eines UB-Baums steht zur Verfügung.

[f)] Problematisch, falls DTDs fehlen.

[g)] Nicht alle Varianten sind leicht updatefähig.

[h)] Hängt von der Implementierung der darunter liegenden Indexstruktur ab (siehe [c)])

Tab. 8-6: *Evaluationszusammenfassung*

Es gibt noch weitere Punkte, die in der Tabelle keinen Platz gefunden haben. Wir werden diese im Folgenden diskutieren. Allgemein lässt sich sagen, dass direkte Vergleiche der Indexstrukturen untereinander im Moment relativ schwierig sind, da aussagekräftige analytische und experimentelle Untersuchungen fehlen. Insofern kann die Beurteilung nur qualitativ und nicht quantitativ erfolgen.

Alle vorgestellten Indexstrukturen unterstützen Wurzelpfadanfragen zumindest für kombinierte Anfragen, während sich für kompliziertere navigierende Anfragen das Feld aufteilt. Volltext-Retrieval wird von den Indexstrukturen teilweise selbst übernommen, teilweise sind andere Komponenten im System dafür vorgesehen (wie z.B. ein Textindex in Lore). Es sollte aber kein Problem sein, die vorgestellten Zugriffsmethoden, bei denen Volltext-Retrieval noch fehlt, um entsprechende Hilfsindexe zu erweitern.

8.6.1 Information-Retrieval-Bereich

Die aus dem Information Retrieval stammenden Ansätze bauen alle auf speziellen Datenstrukturen aus diesem Bereich (wie z.B. invertierte Listen, Tries etc.) auf. Diese Datenstrukturen gehören in den seltensten Fällen zum Standardrepertoire eines relationalen Datenbanksystems und können auch nur ungenügend nachgebildet werden [ZNDL01]. Für den Einsatz dieser Techniken sind gegebenenfalls spezielle Systeme nötig. Die Information-Retrieval-Ansätze sind außerdem relativ gut geeignet für inhaltsbezogene Anfragen mit strukturbezogenen Anteilen. Je navigierender bzw. strukturbezogener die Anfragen werden, desto schlechter wird die zu erwartende Leistung.

Die Leistung eines Kontextfilters steht und fällt mit der False-drop-Rate. Wenn die Anzahl der irrtümlich nicht ausgefilterten Dokumente zu hoch wird, lohnt sich der Mehraufwand mit der Indexstruktur nicht. Kontextfilter berücksichtigen nicht die Positionen und die Anzahl von Vorkommen von Elementnamen in einem Pfad. Wenn man davon ausgeht, dass nicht alle Elementnamen an allen Positionen auftreten und sie meistens nur einmal in einem Pfadausdruck vorkommen, sollten Kontextfilter doch eine Beschleunigung der Anfragegeschwindigkeit bewirken. SEQL bzw. RCS ist genauer als ein Kontextfilter, da keine Schlüsseltransformation (hashing) benutzt wird und die Positionen der Elementnamen zueinander berücksichtigt werden. In der momentanen Form wird nicht zwischen Kindern (direkten Nachfolgern) und indirekten Nachfolgern unterschieden (siehe Tab. 8-2: Umsetzung der Pfadausdrücke A.B und A.B*.C). Dieser Mangel kann allerdings behoben werden, indem für jeden Elementnamen angegeben wird, auf welcher Stufe im Dokument er steht.

8.6.2 Relationale Ansätze

Relationale Ansätze sind gut geeignet, wenn die verwalteten XML-Dokumente nicht zu unstrukturiert sind. Dies lässt sich an der multidimensionalen Indexierung beobachten, die DTDs voraussetzt. Sobald die Ansätze allgemeiner werden,

wie bei XASR/XIS, ist damit zu rechnen, dass die Leistung nachlässt. Im Falle von XASR/XISS wird eine Fülle von Join-Operationen benötigt, um die einzelnen Segmente eines Pfads zusammenzusetzen. Die Ersetzung der Joins durch like-Prädikate verschiebt das Problem in einen Bereich, in dem relationale Datenbanksysteme ebenfalls nicht besonders leistungsfähig sind (eben der Auswertung beliebig komplexer like-Prädikate).

8.6.3 Neue Ansätze

Die neuen Ansätze, obwohl von der Flexibilität und Mächtigkeit her überlegen, haben auch Nachteile. So ist die Komplexität dieser Ansätze wesentlich höher. In Grenzfällen muss mit exponentieller Laufzeit oder exponentiellem Speicherbedarf gerechnet werden. Sie sind auch schwerer zu verstehen und zu implementieren als die anderen Ansätze.

Data Guides können im schlimmsten Fall exponentielle Größe annehmen. Die oberen Schranken von 1-, 2- und T-Indexen sind polynomiell in der Größe des Ursprungsgraphen. So ist der Speicherbedarf eines 1-Index linear, der eines 2-Index quadratisch und der eines T-Index polynomiell n-ten Grades. Ein Problem von T-Indexen gegenüber Data Guides ist allerdings die mangelnde Update-Fähigkeit. Während Data Guides inkrementell geändert werden können, ist dies bei einem T-Index nur möglich, wenn die Schablone auf konstante reguläre Pfadausdrücke beschränkt ist.

8.6.4 Kurzes Fazit

Abschließend lässt sich bemerken, dass keine optimale Indexstruktur auszumachen ist. Alle haben ihre Stärken und Schwächen. Potenzielle Anwender sollten sich dieser Stärken und Schwächen bewusst sein und für eine Anwendung die jeweils geeignete Indexstruktur auswählen.

8.7 Zusammenfassung und Ausblick

Die Indexstrukturen in klassischen Datenbanksystemen sind nur bedingt geeignet, den Herausforderungen der Verwaltung von XML-Daten zu begegnen. Sie scheitern an der effizienten Unterstützung von kombinierten Inhalts- und Strukturanfragen in einer dynamischen Umgebung ohne feste Schemainformation (wie es bei XML-Daten typischerweise der Fall ist). Bei den Neuentwicklungen von XML-Indexstrukturen wurden bereits erste Erfolge im schnellen Verarbeiten von einfachen Pfadausdrücken erzielt, ein großer Durchbruch steht aber noch aus.

Im Bereich von XML-Datenbanksystemen herrschen zwei Strömungen vor. Zum einen gibt es die Integration von XML-Daten und von entsprechenden Indexstrukturen in traditionelle Datenbanksysteme wie relationale Systeme (siehe z.B. [DeFS99, FlKo99, KlMe00, HAGQ97, SKWW00, STZH99, SuRL00,

ZwAW99]), zum anderen die Entwicklung nativer Systeme wie Tamino und Natix [FiKM01, Schö01]. Im Bereich der nativen Systeme ist zu erwarten, dass die Integration neuer Indexstrukturen einfacher ist, da das ganze System auf die Verarbeitung von XML-Dokumenten eingerichtet ist. Eine spannende Frage im Bereich der klassischen Datenbanksysteme bleibt, ob es möglich ist, mit den »Bordmitteln« dieser Systeme leistungsfähige Zugriffsmethoden für XML zu entwickeln oder ob größere Eingriffe in das Laufzeitsystem nötig sind.

Ein direkter Vergleich ist zur Zeit schwierig. Der Grund hierfür ist, dass sich die Entwicklung von Indexstrukturen für XML in einer relativ frühen Phase befindet. So stehen detaillierte Leistungsmessungen und -bewertungen noch aus. Daher werden Indexstrukturen für XML auch in naher und mittlerer Zukunft ein fruchtbares Forschungsgebiet bleiben.

Literatur

[BaRi99] Baeza-Yates, R.; Ribeiro-Neto, B.: *Modern Information Retrieval*. Addison-Wesley, Reading, Massachusetts, USA, 1999.

[Baye01] Bayer, R.: *XML databases: Modeling and multidimensional indexing*. In: DEXA 2001, München, 2001.

[BaCr72] Bayer, R.; McCreight, E. M.: *Organization and maintenance of large ordered indexes*. In: Acta Informatica, Volume1, Number 3, February 1972, S. 173-189. Also published in/as: ACM SIGFIDET 1970, S.107-141.

[Burk92] Burkowski, F.J.: *An algebra for hierarchically organized text-dominated databases*. In: Information Processing and Management, Volume 28, Number 3, 1992, S. 333-348.

[Burk92(2)] Burkowski, F.J.: *Retrieval activities in a database consisting of heterogeneous collections of structured text*. In: Proceedings of the 15th SIGIR Conference, 1992.

[CSFH01] Cooper, B.F.; Sample, N.; Franklin, M.J.; Hjaltason, G.R.; Shadmon, M.: *A fast index for semistructured data*. In: Proceedings of the 27th VLDB, 2001.

[CoSh01] Cooper, B.F.; Shadmon, M.: *The Index Fabric: Technical overview*. Technical report, RightOrder Inc., 2001.

[DeFS99] Deutsch, A.; Fernandez, M.; Suciu, D.: *Storing semistructured data with STORED*. In: SIGMOD Record, Volume 28, Number 2, 1999.

[Naug01] Naughton, J. et al.: *The Niagara internet query system*. In: IEEE Data Engineering Bulletin, Volume 24, Number 2, 2001, S. 27-33.

[FiKM01] Fiebig, T.; Kanne, C.C.; Moerkotte, G.: *Natix – ein natives XML-DBMS*. In: Datenbank-Spektrum, Volume 1, Number 1, 2001, S. 5-13.

[FiMo00] Fiebig, T.; Moerkotte, G.: *Evaluating queries on structure with extended access support relations*. In: WebDB (Informal Proceedings), Dallas, Texas, 2000, S. 41-46.

[FlKo99] Florescu, D.; Kossmann, D.: *Storing and querying xml data using an rdmbs*. In: IEEE Data Engineering Bulletin, Volume 22, Number 3, 1999, S. 27-34.

[GoWi97] Goldman, R.; Widom, J.: *DataGuides: Enabling query formulation and optimization in semistructured databases*. In: 23rd VLDB, 1997, S. 436-445.

[HäRa01] Härder, T.; Rahm, E.: *Datenbanksysteme: Konzepte und Techniken der Implementierung*. Springer, 2001

[KeEi97] Kemper A.; Eickler, A.: *Datenbanksysteme – Eine Einführung*. Oldenbourg, München, 1997.

[Kilp92] Kilpeläinen, P.: *Tree Matching Problems with Applications to Structured Text Databases*. PhD thesis, University of Helsinki, 1992.

[KiMa93] Kilpeläinen, P.; Mannila, H.: *Retrieval from hierarchical texts by partial patterns*. In: Proceedings of the 16th SIGIR Conference, 1993, S. 214-222.

[KlMe00] Klettke, M.; Meyer, H.: *XML and object-relational database systems – enhancing structural mappings based on statistics*. In: ACM SIGMOD Workshop on the Web and Databases (WebDB), 2000.

[LiMo01] Li, Q.; Moon, B.: *Indexing and querying XML data for regular path expressions*. In: Proceedings of the 27th VLDB, 2001.

[HAGQ97] McHugh, J.; Abiteboul, S.; Goldman, R.; Quass, D.; Widom, J.: *Lore: A database management system for semistructured data*. SIGMOD Record, Volume 26, Number 3, 1997.

[HWAL98] McHugh, J.; Widom, J.; Abiteboul, S.; Luo, Q.; Rajaraman, A.: *Indexing semistructured data*. Technical report, Stanford University, Computer Science Department, 1998.

[Meus98] Meuss, H.: *Indexed tree matching with complete answer representations*. In: Fourth Workshop on Digital Document Processing (PODDP'98), 1998.

[MeSt99] Meuss, H.; Strohmaier, C.: *Improving on index structures for structured document retrieval*. In: 21st Annual Colloquium on IR Research (IRSG '99), 1999.

[MiSu99] Milo, T.; Suciu, D.: *Index structures for path expressions*. In: Lecture Notes in Computer Science, Volume 1540, 1999, S. 277-295.

[Morr68] Morrison D. R.: *PATRICIA – practical algorithm to retrieve information coded in alphanumeric*. In: Journal of the ACM, Volume 15, Number 4, October 1968, S. 514-534.

[RMFZ00] Ramsak, F.; Markl, V.; Fenk, R.; Zirkel, M.; Elhardt; K.; Bayer, R.: *Integrating the UB-tree into a database system kernel*. In: 26th VLDB, 2000, pages 263-272.

[SKWW00] Schmidt, A.; Kersten, M.; Windhouwer; M.; Waas, F.: *Efficient relational storage and retrieval of XML documents*. In: ACM SIGMOD Workshop on the Web and Databases (WebDB), 2000.

[Schö01] Schöning, H.: *Tamino – a DBMS designed for XML*. In: ICDE, 2001, S. 149-154.

[SeLK02] Seo, C.; Lee, S.; Kim, H.-J.: *An efficient inverted index technique for xml documents using rdbms*. to be published, http://oopsla.snu.ac.kr/ cyseo/, 2002.

[STZH99] Shanmugasundaram, J.; Tufte, K.; Zhang, C.; He, G.; DeWitt, D.J.; Naughton, J.F. *Relational databases for querying XML documents: Limitations and opportunities*. In: 25th VLDB, 1999, S. 302-314.

[StMe73] Stockmeyer, L.J.; Meyer, A.R.: *Word problems requiring exponential time*. In: 5th STOC, 1973, S. 1-9.

[SuRL00] Surjanto, B.; Ritter; N.; Loeser, H.: *XML content management based on object-relational database technology.* In: Proc. 1st Int. Conf. on Web Information Systems Engineering (WISE), 2000, S. 64-73.

[ZwAW99] Van Zwol, R.; Apers, P.M.G.; Wilschut, A.N.: *Modeling and querying semistructured data with MOA*. In: ICDT '99 Workshop on Query Processing for semistructured data, 1999.

[WiMB99] Witten, I.H.; Moffat; A.; Bell, T.C.: *Managing Gigabytes*. Morgan Kaufmann, San Francisco, 1999.

[ZNDL01] Zhang, C.; Naughton, J.; DeWitt, D.; Luo, Q.; Lohman, G.: *On supporting containment queries in relational database management systems*. In: SIGMOD Record (ACM Special Interest Group on Management of Data), Volume 30, Number 2, June 2001, S. 425-436.

9 Suchmaschinen

Gunnar Weber, Ilvio Bruder, Andreas Heuer

Kurzfassung

Bei der Suche nach Informationen im Web bilden Suchmaschinen neben redaktionell betreuten Katalogen wie Yahoo! die zentralen Einstiegspunkte. Die Grundlagen für Suchmaschinen stammen aus dem Gebiet des Information Retrieval (IR) und werden im ersten Teil des Kapitels vorgestellt. Der zweite Teil beschäftigt sich dann mit der Anpassung dieser Techniken an die Erfordernisse des World Wide Web und stellt gängige Suchmaschinen-Architekturen sowie aktuelle Systeme vor. Web-Suchmaschinen beschränken sich auf die Suche von Textinhalten, die Struktur der Dokumente wird nicht berücksichtigt. Das Potenzial, das in solchen Strukturen und deren Interpretation für eine effektivere Suche steckt, soll im dritten Teil diskutiert und anhand von aktuellen XML-Suchmaschinen bzw. erweiterten XML-Anfragesprachen illustriert werden. Einen weiteren Schwerpunkt bildet die Kombination von Datenbank- und IR-Techniken zum Durchsuchen semistrukturierter Dokumente.

9.1 Einleitung

Suchmaschinen findet man in sehr vielen Systemumgebungen, im Unternehmensnetz als Informationsserver, in Bibliotheken zur Suche in elektronischen Dokumenten oder als Web-Dienst im Internet. Die Suchmaschinen im Internet sind dabei sicherlich die bekanntesten. Wie sieht nun die Technik hinter den Suchmaschinen aus und wie finden diese die relevanten Information, die ich suche? Solche und weitere Fragen sollen in diesem Kapitel diskutiert werden.

Suchmaschinen werden i.A. mit Information Retrieval in Verbindung gebracht, wobei im herkömmlichen Sinne Systeme gemeint sind, die eine Suche auf Dokumenten anbieten, ohne diese auch geeignet zu verwalten. Demnach werden bei einer Suchmaschine Eigenschaften und Charakteristika von Dokumenten gesammelt. Diese Dokumentbeschreibungen können über relativ einfache Suchanfragen durchsucht werden. Suchtreffer enthalten dann eine Referenz auf die zugehörigen Dokumente. Information Retrieval ist gewöhnlich die vage Suche auf Dokumentinhalte und deren unspezifische Bewertung. Im Gegensatz dazu spricht man bei Datenbanken von Fakten-Retrieval. In Datenbanken können hauptsäch-

lich exakte Anfragen an Daten mit spezifizierter Semantik gestellt werden. Weitere Betrachtungen zur Unterscheidung der Techniken finden sich u.a. in [VR79].

Jedoch wird der Begriff Suchmaschine oftmals in viel umfangreicherem Kontext verwendet. Das liegt nicht zuletzt daran, dass mehr und mehr die verschiedenen Anfrage-, Such- und Retrievaltechniken bzw. strukturiertes und unstrukturiertes Retrieval integriert werden.

Die Basistechniken für Suchmaschinen liefern die herkömmlichen Information-Retrieval-Systeme. Erweiterte und aktuelle Techniken passen diese Retrieval-Techniken an bestimmte Umgebungen wie das Web oder an bestimmte Dokumenttypen wie HTML oder XML an. Der Strukturierungsgrad ist ein signifikantes Merkmal eines Dokumenttyps. Während herkömmliche Information-Retrieval-Systeme hauptsächlich von unstrukturierten Texten (Volltexten) ausgehen, werden in diesem Kapitel auch Suchmöglichkeiten auf semistrukturierten Dokumenten betrachtet (z.B. Volltext mit definierten Abschnitten und Metadaten wie Autor, Titel).

In diesem Sinne werden Suchmaschinen unter folgenden Gesichtspunkten diskutiert:

- *Information-Retrieval-Techniken:* Anhand der etablierten Methoden von Text-Retrieval werden die Grundtechniken von Suchmaschinen in Abschnitt 9.2 erläutert. Dazu gehören die Deskribierung der Texte, die Recherchetechniken und die Bewertung der Ergebnisse.
- *Web-Suchmaschinen:* Im Abschnitt 9.3 werden die Information-Retrieval-Techniken an die Web-Charakteristik angepasst. Es werden die speziellen Anforderungen diskutiert und verschiedenartige Ansätze klassifiziert. Anschließend werden Architekturkonzepte für Web-Suchmaschinen vorgestellt. Es wird hier prinzipiell zwischen zentralisierter und verteilter Architektur unterschieden. Zum Ende dieses Abschnitts werden konkrete Systeme vorgestellt.
- *Suchmaschinen auf semistrukturierten Dokumenten:* Abschnitt 9.4 illustriert zunächst Möglichkeiten zur Verschmelzung von Datenbank- und IR-Techniken für eine praktikable Suche auf semistrukturierten Daten. Eine mögliche Kombination beider Techniken wird anhand des Sprachvorschlags IRQL betrachtet. Anschließend erfolgt die Vorstellung erweiterter XML-Anfragesprachen, die fehlende IR-Fähigkeiten in aktuelle XML-Sprachvorschläge integrieren. Web-Suchmaschinen beschränken sich in ihrem Retrieval auf die Auswertung von Wort- bzw. Termhäufigkeiten und die Analyse von Linkstrukturen zwischen Dokumenten. Das Potenzial, das z.B. in den Strukturen bzw. semantischen Annotationen von XML-Dokumenten für eine effektivere Suche steckt, bleibt völlig ungenutzt. Aus diesen Überlegungen sind XML-Suchmaschinen entstanden, die am Ende dieses Abschnitts ebenfalls betrachtet werden.

9.2 Einführung in Information-Retrieval-Techniken

Information-Retrieval-Systeme für Texte oder unstrukturierte Daten sind bereits länger als Datenbanksysteme verfügbar. Die Grundtechniken wurden schon in den sechziger Jahren entwickelt und teilweise auch in Systemen eingesetzt. Somit sind Information-Retrieval-Systeme deutlich früher entstanden als relationale Datenbanksysteme (die damals unter dem Begriff Fakten-Nachweissysteme eingeordnet wurden).

Diese Techniken bilden die Grundlage für Suchmaschinen und für den Datentyp `Text` in heutigen objektrelationalen Systemen.

Grundprinzip und Ziel des Information Retrieval

Im Gegensatz zu klassischen Datenbankanfragen sind Anfragen an Texte in der Regel keine scharfen Anfragen, die das Anfrageergebnis eindeutig mit »Ja«- und »Nein«-Entscheidungen aufbauen. Stattdessen erfolgt ein »Ranking« der Dokumente aufgrund des Grades der Übereinstimmung mit dem Suchkriterium.

Die Effektivität von Information-Retrieval-Techniken wird durch die Kennzahlen *Recall* und *Precision* bewertet:

- Mit *Recall* wird die Anzahl relevanter Objekte im Ergebnis im Verhältnis zur Anzahl aller relevanten Objekte gemessen. Ein hoher Recall beschreibt eine Suchmethode, die eine große Anzahl der gemäß Suchkriterium qualifizierten Dokumente findet.
- Die *Precision* beschreibt die Anzahl relevanter Objekte im Ergebnis im Verhältnis zur Anzahl aller Objekte im Ergebnis. Eine hohe Präzision bedeutet, dass wenig »Datenmüll« gefunden wird, also Dokumente, die irrtümlich als Treffer qualifiziert wurden.

Grundtechniken

Die Grundtechniken des Information Retrieval können in drei Phasen aufgeteilt werden:

- *Deskribierung:* Die Deskribierung beschreibt allgemein die Transformation eines Textdokumentes in eine Dokumentbeschreibung. Dazu gehören zum einen Beschreibungen aufgrund von Metadaten, wie etwa die bibliographischen Angaben aus einem Zeitschriftenartikel (Autoren, Band, Heft und Seitenzahl), und zum anderen so genannte Terme, die Schlagworte über den Text oder Stichwörter aus dem Text darstellen.
- *Recherche:* Hierunter versteht man das Suchen von Textdokumenten nach Vorgabe von Termen, Metadaten und Operatoren.
- *Bewertung:* Eine Bewertung ordnet die gefundenen Dokumente ihrer Güte bzgl. der gestellten Recherche nach an und ermöglicht eine Reaktion des Nutzers.

Die Techniken der Deskribierung, Recherche und Bewertung werden nun im Einzelnen skizziert.

9.2.1 Deskribierung

Die Verfahren zur Deskribierung kann man aus verschiedenen Blickwinkeln betrachten:

- Die *statistischen, wortbasierten Verfahren* sind im Bereich der Suchmaschinen am gebräuchlichsten. Diese Verfahren analysieren den Text aufgrund von Worten, die in ihm vorkommen (dieser Vorgang wird unten noch mit *Indexierung* bezeichnet). Es werden dabei nur sehr einfache linguistische Analysen vorgenommen, etwa die Beugungsformen eines Wortes bei Deskribierung und Recherche berücksichtigt. Eine Bewertung der Texte geschieht oft nach statistischen Maßzahlen wie die Häufigkeit des Wortes im Text.
- Die *linguistischen Verfahren* analysieren Worttypen und Satzstrukturen der Texte und versuchen, die Worte im Satzzusammenhang zu erkennen. Dazu ordnen sie den Worten bestimmte linguistische Rollen im Satz zu (etwa: Subjekt, Eigenname, Nominalphrase, Präpositionalphrase).
- Die *wissensbasierten Verfahren* versuchen, mit Hilfe von Ontologien das Anwendungsgebiet zu strukturieren. Ontologien kann man sich wie ein objektorientiertes Datenbankschema für ein bestimmtes Anwendungsgebiet vorstellen, in dem die Begriffe der Anwendung in verschiedene Spezialisierungs- oder Komponentenhierarchien eingeordnet werden (Paperback ist spezieller als Buch, Kapitel ist Teil eines Buches).

Die folgenden Erläuterungen werden sich auf die verbreiteten wortbasierten Verfahren konzentrieren. Eine Kombination aller drei Techniken zur Textanalyse ist bspw. in [Kl+01] zu finden.

Indexierung mit Stichwortverfahren

Bei der Indexierung von Texten sind die Terme entweder Worte aus dem Text (wie bei Stichwortverfahren, die oft nur Worte aus einem Glossar oder einer Positivliste zulassen) oder Worte über den Text (wie bei Schlagwortverfahren, die ebenfalls nur Terme aus einem kontrollierten Wortschatz zulassen).

Ein Thesaurus kann zu Stichwörtern oder Schlagworten noch Vorzugsbenennungen, Querverweise, Ober- und Unterbegriffe, *Benutzt-für*-Beziehungen, Synonyme, Antonyme oder weitere Bezüge zu anderen Begriffen speichern.

Man kann bestimmte Terme noch attributieren (auch Aspekte oder Rollen genannt). Beispielsweise kann für den Begriff »Müller« die Attributierung `Name: Müller` oder `Beruf: Müller` festgelegt werden.

Das Standardverfahren zur Indexierung ist das Stichwortverfahren. Das Stichwortverfahren extrahiert aus Dokumenten diejenigen Wortformen, die zur

Suche eingesetzt werden können und daher in einer Zugriffsstruktur verwendet werden. Hier kann wie folgt vorgegangen werden:

- Es wird die Häufigkeit aller Worte in den Texten bestimmt.
- Die *häufigsten Worte* werden gestrichen, da Worte wie »er«, »ein«, »oder« keinen Beitrag zur Bestimmung der Relevanz von Dokumenten bringen. Statt die häufigsten Worte zu streichen, kann auch eine Liste von so genannten *Stoppworten* eingesetzt werden. Die Worte aus der Stoppwortliste (Negativliste) werden dann aus dem Index gestrichen.
- Manchmal werden auch die *seltensten Worte* gestrichen, obwohl diese ja die maximale Selektivität haben. Der Grund ist, dass seltene Worte meist nur einen kleinen Teil des Dokuments beschreiben und solche Worte außerdem zu stark selektieren, so dass einige relevante Dokumente bei einer dementsprechenden Anfrage unberücksichtigt bleiben.
- Von den verbleibenden Worten werden oft nur die Stammformen in den Index übernommen. Für die Flexionsformen, d.h. die Deklination von Substantiven und Adjektiven, die Konjugation von Verben und die Komposition von Worten, wird ein linguistischer Index angelegt, der sowohl bei der Deskribierung als auch bei der Recherche verwendet wird.

Nachdem die Aufbereitung der Dokumente durch eine Deskribierung erfolgt ist, können Anfragen an die Menge der Textdokumente gestellt werden. Da man Anfragen an Texte unscharf und nicht mit Attribut-Wert-Vergleichen und Verbundoperationen formuliert, werden Anfragen an Texte im Folgenden auch Recherche-Operationen genannt.

9.2.2 Recherche

Übliche Anfragen an Texte basieren auf den in ihnen enthaltenen Wörtern (Stichwörtern) oder den festgelegten Schlagworten. Bei *einfachen Recherchen* werden ausschließlich Terme angegeben und diese in vielfältiger Weise mit Operatoren, Kontextinformationen oder Wichtungen versehen. Folgende Typen von einfachen Recherchen gibt es:

- *Angabe eines Terms bzw. einer Termmenge:* Das ist die einfachste Rechercheform, bei der als Angabe nur ein Term erlaubt ist. Solche Recherchen entsprechen Anfragen auf das Enthaltensein eines Wortes in einem Text:

  ```
  [ 'Datenbanken' ] in Text
  ```

  Bei großen Textdatenbanken stoßen diese einfachen Anfragen schnell an ihre Grenzen, da zu viele Texte das Suchwort enthalten können. Wird eine Termmenge angegeben, dann sollen alle Terme im gesuchten Dokument vorkommen. Die resultierenden Ergebnisdokumentmengen pro Term werden also geschnitten.

- *Boolesche Recherchen:* Dies ist die üblichste Form einer Recherche. Verschiedene Terme werden durch *and, or* und *not* verknüpft. *Konjunktive Anfragen* bestimmen beispielsweise Texte, die vorgegebene Suchbegriffe gemeinsam enthalten:

  ```
  [ 'Datenbanken' and 'Sprachen' ] in Text
  ```

  Ein einfaches *not* `Term` ist nicht erlaubt, da Negationen in der Regel nur in konjunktiven Verknüpfungen hinreichend spezifische Suchkriterien ergeben. Die Ergebnisdokumentmengen werden mit den Mengenoperationen $\cup$, $\cap$ und - verknüpft.

- *Kontext-Recherchen:* Hier wird ein Kontext für die jeweiligen Suchterme angegeben, beispielsweise eine absolute Ortsangabe (in einem bestimmten Kapitel) oder eine relative Ortsangabe (dieser Term drei Abschnitte vor dem anderen Term).

  ```
  [ 'Datenbank' 1 Wort vor 'Sprache' ] in Text,
  [ 'Datenbank' im gleichen Satz mit 'Sprache' ] in Text,
  [ 'Datenbank' innerhalb 2 Abschnitte mit 'Sprache' ] in Text
  ```

  Das Textdokument muss dazu in entsprechender Weise (Kapitel, Abschnitte, Sätze) strukturiert sein. Der Index muss die Ortsinformationen ebenfalls berücksichtigen. Übliche metrische Operationen sind *gleicher Satz*, *gleiche Struktureinheit* (wie Abschnitt, Kapitel), *Abstand zum Satz* oder *zur Struktureinheit*, vorgegebener *Abstand zwischen Termen*, etwa auch in Anzahl von Worten.

  Die einfachste Form der Kontext-Recherche ist die Suche nach einer Phrase, bei der die gesuchten Terme genau in dieser direkten Folge im Text auftauchen sollen.

- *Gewichtete Recherchen:* Unwichtigere Terme können hier von wichtigen unterschieden werden, indem jedem Term ein Wichtungsfaktor mitgegeben wird. Kommen z.B. bei einer mit *and* verknüpften Termmenge nicht alle Terme in einem Dokument vor, so kann man mit der Gewichtung Dokumente höher bewerten (s.u. Vektorraummodell und Ranking), die mehr und wichtigere Terme enthalten. Beispielsweise kann mit der gewichteten Anfrage

  ```
  Datenbank: 0.8 and Sprache: 0.5 and Konzepte: 0.2
  ```

  eine Publikation über Datenbanksprachen, aber ohne Vorstellung von Konzepten, höher eingestuft werden als eine Publikation über Sprachkonzepte ohne Hinweis auf Datenbanken.

- *Suche nach Mustern:* Schließlich können auch die gegebenen Terme selbst nur durch Muster beschrieben sein. So möchte man bei dem Term `Datenbank` manchmal exakt dieses Wort, manchmal nur den Präfix, sehr oft aber eine beliebige Teilzeichenkette auch innerhalb eines Wortes finden. Oft soll die Groß- und Kleinschreibung dabei unberücksichtigt bleiben (und somit für

obigen Term auch Dokumente mit dem Wort `Objektdatenbank` gefunden werden).

Weitere Ungenauigkeiten können das Zulassen von x Fehlern in einem Suchbegriff sein (so findet man für den Suchbegriff `Schek` auch das Wort `Scheck` oder beim Suchbegriff `daß` auch die Terme `das` und `dass`).

Komplexere Recherchen nehmen nun zusätzlich auf die Strukturierung des Dokumentes Rücksicht. Erste Ansätze dazu wurden eben mit den Kontext-Recherchen schon vorgestellt. Weitere Möglichkeiten wären eine *Attributierung* (falls diese in der Struktur des Dokumentes durch eine semantische Auszeichnung bekannt ist, wie etwa in XML). So kann man beispielsweise die obige Anfrage nach Datenbank-Publikationen mit den attributierten Termen `title: Sprache` und `chapter: Konzepte` semantisch bereichern.

Sind im Textdokument solche Attribute nicht ausgezeichnet, können sie entweder durch manuelle Deskribierung erfasst (wie in Bibliotheksanwendungen üblich) oder durch automatische Deskribierungsverfahren ermittelt werden.

Retrieval-Modelle

Die mit einer der obigen Operationen formulierten Recherchen können nun nach unterschiedlichen Modellen ausgewertet werden:

- *Boolesches Retrieval*[1]: Das boolesche Retrieval liefert *true*, wenn die Terme im Dokument vorkommen, und *false*, wenn sie nicht vorkommen. Werden aufgrund einer Recherche mehrere Dokumente zurückgeliefert, so haben diese Dokumente die gleiche Relevanz.
- *Vektorraummodell*: Die verschiedenen Such- und Dokument-Terme können jeweils als ein Vektor in einem mehrdimensionalen Raum aufgefasst werden. Auch wenn die Dokument-Terme nicht genau die Suchterme treffen, so können mit einem Ähnlichkeitsmaß Dokumente mit »benachbarten« Vektoren als Ergebnis zurückgegeben werden. Das Ergebnis ist in diesem Fall vage. Weiterhin können noch Gewichte für Such- und Dokument-Terme vergeben werden, die die Ähnlichkeit der Vektoren noch weiter beeinflussen. Beispiele für ein auf diesem Modell basierendes Ranking-Verfahren werden wir weiter unten noch präsentieren.
- *Probabilistisches Modell*: Im Gegensatz zu Wichtungen und Ähnlichkeitsmaßen werden in diesem Modell Wahrscheinlichkeiten berechnet, die aussagen, ob ein Dokument bezüglich der Recherche relevant ist oder nicht.

1. Das boolesche Retrieval-Modell ist nicht zu verwechseln mit der booleschen Recherche: In der booleschen Recherche werden Terme mit *and*, *or* und *not* verknüpft, im booleschen Retrieval-Modell als Ergebnis einer Recherche für ein Dokument nur **true** oder **false** geliefert.

9.2.3 Ranking und Relevance Feedback

Eine Bewertung der gefundenen Dokumente kann im Information Retrieval in zwei Phasen erfolgen. Zum einen werden die gefundenen Dokumente in der Reihenfolge ihrer Relevanz mittels einer Ranking-Funktion ausgegeben. Zum anderen kann beim Relevance Feedback der Nutzer interaktiv eingreifen und bestimmte Dokumente als sehr relevant oder irrelevant markieren.

Ranking

In einer Recherche gefundene Dokumente sollen nun gemäß ihrer Relevanz absteigend sortiert werden. Dazu ist zu berechnen, wie relevant das Dokument ist. Als Voraussetzung dazu muss zunächst die Relevanz eines oder mehrerer Terme für ein Dokument bestimmt werden.

Die grundlegenden Maße dabei sind:

- M: Die Anzahl der Terme, $m \leq M$ stellt einen beliebigen Term dar
- N: Die Anzahl der Dokumente in der Datenbasis, $n \leq N$ ist ein beliebiges Dokument aus der Datenbasis
- f_{mn}: Die Häufigkeit des Terms T_m im Dokument D_n
- d_m: Die Anzahl der Dokumente in der Datenbasis, in denen der Term T_m auftritt
- $idf_m(N,d_m)=log(N/d_m)$ = Inverse Dokumenthäufigkeit (IDF - inverse document frequency) für einen Term T_m

Die Anwendung der zugrunde liegenden Maße ist abhängig von den eingesetzten Heuristiken. Die Bewertung w_{nm} des Terms T_m in Dokument D_n wird dann aufgrund der obigen Parameter und der Heuristiken ermittelt. Die Tabelle 9-1 gibt eine kleine Auswahl von möglichen Heuristiken und zugehörigen Formeln.

Ranking-Formel	Heuristik
$1/d_m$	Ein im Dokument D_n enthaltener Term T_m, der nicht so häufig in der Datenbasis auftritt, ist wichtiger als andere, oft auftretende Terme.
f_{mn}	Die Häufigkeit des Terms T_m in einem Dokument D_n ist entscheidend.
$f_{mn} \cdot idf_m$	Je häufiger ein Term T_m im Dokument D_n auftaucht, aber je weniger Dokumente diesen Term beinhalten, um so relevanter ist der Term.

Tab. 9-1: *Ranking-Verfahren für einen Term und ein Dokument*

Die Gesamtrelevanz R_n des Dokuments D_n zu einer Anfrage Q berechnet sich in Abhängigkeit des verwendeten Retrieval-Modells.

Im *booleschen Modell* spielen die Ranking-Funktionen keine Rolle. Ein Dokument ist im Ergebnis enthalten, wenn das boolesche Prädikat, das die Recherche darstellt, **true** ergibt (siehe auch Abbildung 9.1a).

Beim *Vektorraummodell* werden die Gewichte für ein Dokument D_n in einem Vektor $\vec{D}_n = (w_{1n},...,w_{Mn})$ definiert. Für die Anfrageterme wird ebenfalls ein Vektor

$\vec{Q}_n = (w_{1q},...,w_{Mq})$ gebildet. Zur Relevanzberechnung kann nun ein Ähnlichkeitsmaß zwischen dem Dokumentvektor und dem Anfragevektor als Kosinus des Winkels definiert werden (siehe Abbildung 9.1b). Der Winkel berechnet sich aus den Vektoren folgendermaßen:

$$R_n = \frac{\vec{D}_n \times \vec{Q}}{\left|\vec{D}_n\right| \cdot \left|\vec{Q}\right|} = \frac{\sum_{m=1}^{M} w_{mn} \cdot w_{mq}}{\sqrt{\sum_{m=1}^{M} w_{mn}^2} \cdot \sqrt{\sum_{m=1}^{M} w_{mq}^2}}$$

Die Ähnlichkeitsmaße zwischen Anfrage und Dokument werden absteigend sortiert und bilden die nach Relevanz geordnete, resultierende Dokumentliste.

Das p*robabilistische Modell* bewertet das Ranking nach der Wahrscheinlichkeit, dass das Dokument D_n relevant für die Anfrage ist (siehe Abbildung 9-1c). Dabei wird die Menge der Dokumente D in Relevante R und nicht Relevante $\overline{R}$ aufgeteilt, wobei R komplementär zu $\overline{R}$ ist. Das Ranking kann nun aus den Wahrscheinlichkeiten $P(R|\vec{D}_n)$ (D_n ist relevant) und $P(\overline{R}|\vec{D}_n)$ (D_n ist nicht relevant) berechnet werden.

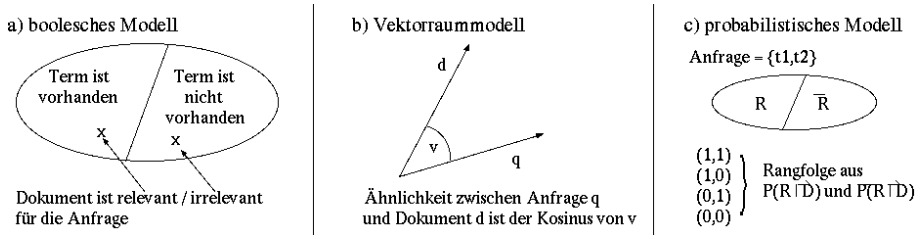


Abb. 9-1: *Veranschaulichung des Ranking im booleschen Modell, im Vektorraummodell und im probabilistischen Modell*

Bei den existierenden Web-Suchmaschinen und Indexierungswerkzeugen wird vorrangig das Vektorraummodell oder ein erweitertes boolesches Modell (boolesches Modell plus Ranking-Werte) eingesetzt. Abschnitt 9.3.5 wird die hier genannten Techniken aufgreifen und an Web-Suchmaschinen anpassen.

Relevance Feedback

Die nach dem Ranking-Verfahren bewertete und sortierte Liste der Ergebnisdokumente kann nun vom Nutzer mit dem interaktiven Verfahren des Relevance Feedback (Relevanzrückkopplung) verbessert werden. Allgemeine Idee des Relevance Feedback ist es, dem System die relevanten und irrelevanten Dokumente aus der Ergebnisliste zu nennen, so dass das IR-System in einem zweiten Anlauf seine Anfrage konkretisieren oder den Nutzerwünschen entsprechend anpassen kann.

Eine weitere Möglichkeit ist die Einbeziehung von Termen aus einem Thesaurus, die für jeden Recherche-Term interaktiv oder automatisch ausgewählt und in der Anfrage zusätzlich berücksichtigt werden können.

Insgesamt soll die weiter oben angesprochene Selektionsgüte wie die Relevanzquote (Precision) oder die Nachweisquote (Recall) dadurch verbessert werden.

9.3 Web-Suchmaschinen

Suchmaschinen wie Google oder Altavista nutzen eine Reihe der genannten IR-Techniken. Im Folgenden werden zuerst die Web-Charakteristika und die daraus resultierenden Anforderungen an Web-Suchmaschinen diskutiert, anschließend werden Architekturen und Techniken vorgestellt. Am Ende dieses Abschnitts erfolgt die Beschreibung konkreter Systeme.

9.3.1 Information Retrieval im Web

Das Web ist ein über Links verbundenes Netz von Dokumenten, das zugegriffen, navigiert und durchsucht werden kann. Die Dokumente werden auf Web-Servern gehalten, die Verwaltung geschieht mittels einfacher Dateistrukturen oder Datenbanken. Die Dokumente gehören zu unterschiedlichen Dokumenttypen (meist HTML) und bestehen aus verschiedenen Multimedia-Elementen. Die Anzahl der über eine eindeutige URL direkt addressierbaren Dokumente steigt ständig – 1999 ca. 800 Millionen, Anfang 2000 über 1 Milliarde und Ende 2000 ungefähr 2,5 Milliarden Dokumente [LaGi99, Berk00]. Dokumente, die in Datenbanken, Katalogen oder anderen dynamisch generierten Web-Informationssystemen durch eine gemeinsame URL mit unterschiedlichen Parametern ansprechbar sind, werden ins so genannte »Hidden Web« eingeordnet.

Die Werte für die Parameter kann der Nutzer im Allgemeinen in HTML-Formularen angeben und anschließend an den Web-Server senden, der als Ergebnis ein entsprechendes Dokument zurückliefert. Schätzungen zur Größe dieses WWW-Bereichs gehen vom 150fachem bis 550fachem des direkt adressierbaren Web aus [Berk00, Berg01]. Diese enorme Anzahl von Web-Dokumenten sind auf ungefähr 8,5 Millionen Web-Servern (1999 4,5 Millionen) verteilt und beanspruchen bis zu 50 Terabyte Daten (1999 etwa 8 Terabyte) [OCLC01, LaGi99, Berk00]. Suchmaschinen sind die meist besuchten Webseiten im Internet und werden von rund 85% der Internet-Anwender genutzt.

Aus den genannten Fakten ergeben sich spezielle Anforderungen an Suchmaschinen und deren Retrieval-Technik.

- *Abdeckung:* Alle im Web verfügbaren Informationen sollen eingesammelt werden und somit über Suchmaschinen auffindbar sein. Diese Forderung ist zur Zeit leider nur ein Wunsch. Alle aktuell verfügbaren Suchmaschinen zusammen sammeln momentan nur rund 60% der im WWW direkt erreichbaren Dokumente ein. Das Problem besteht hauptsächlich in der zwar gro-

ßen, aber begrenzten Skalierbarkeit der Indexierungs- und Retrieval-Techniken durch Netzbandbreite sowie Hard- und Software-Performance. Der technische Aufwand für das Indexieren ist bei den oben genannten Zahlen enorm.

Nicht eingesammelt werden nicht bekannte bzw. verlinkte Server, explizit gesperrte bzw. durch Firewalls geschützte Dokumente, dynamisch generierte Webseiten, Web-Datenbanken sowie die in speziellen Konfigurationsdateien angegebenen Dokumente (Robot-Exclusion-Standard).

- *Verfügbarkeit:* Suchmaschinen sind global verfügbar und sollten rund um die Uhr erreichbar sein. Google bspw. erhält täglich 130 Millionen Anfragen [IMG01]. Schon Ausfälle von nur wenigen Minuten fallen sehr schnell auf.

  Die Verfügbarkeit gilt nicht nur im Zusammenhang mit Nutzeranfragen, auch beim Sammeln wirken sich Ausfälle schnell negativ auf die Aktualität des Angebots aus.

- *Aktualität:* Die Aktualität des Indexes ist aufgrund der Dokumentmenge im WWW ein schwieriges Problem. Die durchschnittliche Aktualität der Suchmaschinen liegt zwischen 50 und 150 Tagen.

- *Nutzerfreundlichkeit:* Anfrageschnittstellen sollten intuitiv benutzbar sein, da bei allgemeinen Suchmaschinen das Nutzerwissen nicht eingeschätzt werden kann. Außerdem ist ein Eingreifen des Nutzers in den Ranking-Prozess oftmals hilfreich. Dies kann durch Relevance Feedback oder durch direktes Modifizieren der Anfragegewichte geschehen.

  Aktuelle Suchmaschinen bieten neben einfachen Anfragemöglichkeiten auch komplexere Anfragemethoden mit erweiterten Such- und Ranking-Einstellungen an.

- *Ergebnispräsentation und Ranking:* Das eben schon angesprochene Ranking ist der wichtigste Aspekt der Ergebnispräsentation. Dazu kommt die eigentliche Dokumentpräsentation innerhalb der Ergebnismenge, die neben der URL diverse weitere Informationen wie Aktualität und Dokumenttyp sowie eventuell eine kurze Zusammenfassung des Dokumentinhalts umfasst.

Im WWW werden vorrangig Information-Retrieval-Techniken angewendet, die an die Gegebenheiten und Charakteristika des Web angepasst sind. Zu den in Abschnitt 9.2 genannten Information-Retrieval-Techniken kommen weitere speziell bei Internet-Suchmaschinen auftretende Aspekte hinzu.

Während beim klassischen Information Retrieval von bestehenden Dokumentkollektionen ausgegangen wird, müssen bei Suchmaschinen die Dokumente im WWW eingesammelt werden. Dazu werden die Link-Referenzen zwischen den Dokumenten ausgenutzt, um von bekannten Dokumenten zu weiteren, bisher nicht eingesammelten Dokumenten zu kommen. Die dabei ermittelten Linkstruk-

turen können zur Verbesserung des Rankings der Ergebnisdokumente genutzt werden (siehe auch 9.3.5).

Eine weitere Herausforderung im WWW ist die enorme Vielfalt der Dokumenttypen, die von strukturierten Informationen bis hin zu unstrukturierten Dokumenten reicht. Hierbei spielt zwar der HTML-Typ nach wie vor die Hauptrolle, andere Dokumenttypen finden allerdings zusehends mehr Verbreitung (in erster Linie XML).

Klassifikation von Suchmaschinen

Nach allgemeinen Betrachtungen zu klassischen Internet-Suchmaschinen sollen nun die verschiedenen Arten vorgestellt werden. Die im Web auftretenden Suchmaschinen lassen sich wie folgt klassifizieren:

- *Crawler-basierte Suchmaschinen*: Verwendung im WWW und großen Dokumentnetzen, aber auch lokal
  Bsp: Altavista, Lycos, Webcrawler, Google, Inktomi
- *Metasuchmaschinen*: Verwendung im WWW
  Bsp: MetaGer, MetaCrawler
- *Spezialisierte Suchmaschinen*: Verwendung in Dokumentnetzen aller Größenordnungen, meist auf bestimmte Teilmenge von Dokumenten zugeschnitten
  Bsp: GETESS, MP3-Search, SWING
- *Verzeichnisdienste, Kataloge*: Verwendung in Dokumentnetzen aller Größenordnungen, meist auf bestimmte Teilmenge von Dokumenten zugeschnitten
  Bsp: Yahoo!, LookSmart

Die einzelnen Suchmaschinenarten werden nun anhand der Kategorien Sammelprozess, Analyse, Indexierung, Anfragemöglichkeiten und Ergebnispräsentation genauer vorgestellt.

Crawler-basierte Suchmaschinen

Das Einsammeln der Dokumente erfolgt durch die Definition von Startpunkten und dem Weiterfolgen von Links zu weiteren Web-Dokumenten. Dieses Sammeln nennt man auch Crawlen und das Werkzeug Crawler. Eine dokumentspezifische Analyse liefert zum einen Links für den Sammelprozess und zum anderen Daten für die Indexierung. In den meisten Fällen wird der Volltext des Dokuments indexiert. Teilweise unterstützen einige Suchmaschinen auch eine attributierte Speicherung von ausgezeichneten Textbestandteilen. Die Aktualisierung der Daten erfolgt durch erneutes Einsammeln, die erneute Indexierung eines Dokuments muss nur bei Veränderung des Inhalts vorgenommen werden. Nicht mehr vorhandene Dokumente sind aus dem Index zu löschen. Anfragen werden hauptsächlich als inhaltsbasiertes Retrieval umgesetzt, eine attributierte Suche ist vom Aufbau des Indexes abhängig. In der Regel werden boolesche Verknüpfungen von Suchwörtern zugelassen. Die Bewertung der Ergebnisse (Ranking) erfolgt grundsätz-

lich nach den bekannten IR-Methoden. Dazu beeinflussen weitere in Abschnitt 9.3.5 vorgestellte Wertungsgewichte die letztendlichen Ranking-Zahlen.

Meta-Suchmaschinen

Mit Meta-Suchmaschinen können Internet-Nutzer mehrere Suchmaschinen gleichzeitig anfragen und damit die Abdeckung erhöhen. Der Sammelprozess entfällt somit, da die Daten bestehender Suchmaschinen genutzt werden. Konkret werden zur Anfragezeit eine gewisse Anzahl von Suchmaschinen befragt und deren beste Ergebnisse nach deren Ranking gesammelt. Die Anfrage erfolgt über eine normierte Suchschnittstelle über die verwendeten Suchmaschinen, so dass die grundlegenden Anfragemöglichkeiten (die alle verwendeten Suchmaschinen gemeinsam anbieten) gegeben sind. Allerdings verliert man die Möglichkeit, Suchmaschinen-spezifische Eigenschaften auszunutzen. Ein wesentliches Problem von Meta-Suchmaschinen ist die Vereinheitlichung der Ergebnislisten. Dazu gehört neben der einheitlichen Darstellung der Ergebnisse auch ein einheitliches Ranking sowie das Eliminieren von Duplikaten (bspw. über die URL).

Spezialisierte Suchmaschinen

Spezialisierte Suchmaschinen sind im Allgemeinen Spezialfälle der Crawler-basierten Suchmaschinen. Eine Spezialisierung ermöglicht die Anpassung der Analysen, Speicherungsformen und Anfragefunktionalitäten an die Gegebenheiten des jeweiligen Spezialgebiets. Beispielsweise kann eine Suchmaschine für Informatikpublikationen die Nutzerschnittstelle sowie den Sammel- und Indexierungsprozess an die Informatikterminologie anpassen. Folgende Spezialisierungen bzgl. des Sammelprozesses sind möglich: Spezialisierung auf einem geographischen Gebiet (z.B. SWING [Webe01]), auf bestimmte Web-Server (bspw. lokale Suchmaschinen), auf eine bestimmte Thematik oder Wissensdomäne (GETESS [Kl+01]) oder auf spezielle Dokumenttypen (etwa MP3-Search). Man erhofft sich bessere Analyse- und Indexierungstechniken und in der jeweiligen Spezialität eine quantitativ oder qualitativ höhere Abdeckung.

Verzeichnisdienste, Kataloge

Verzeichnisdienste bzw. Kataloge sind nicht immer klar als Suchmaschine identifizierbar. In einigen Fällen werden auch datenbankgestützte Informationssysteme verwendet und damit auch eine direkte Verwaltung der Dokumente vorgenommen. Der Sammelprozess wird hier in der Regel von einem Team von Sachverständigen geführt, wobei rechnergestützte Kategorisierungswerkzeuge mitgenutzt werden. Neben dem Sammeln und Katalogisieren werden die Webseiten auch analysiert und bewertet. Das Problem dieser Methode ist die sehr geringe Abdeckung gegenüber einem hohen Aufwand (Personal). Dafür sprechen aber gute und umfangreiche Anfragemöglichkeiten und gute Anfrageergebnisse (hohe Precision).

9.3.2 Typische Suchmaschinen-Architekturen

Die meisten Suchmaschinen haben eine zentralisierte Crawler-Indexierer-Architektur, die in Abbildung 9-2 zu sehen ist [BaRi99]. Zentralisiert bedeutet, dass Crawler und Indexierer eng miteinander gekoppelt sind, d.h., die vom Crawler ermittelten Daten werden nicht zwischengespeichert, sondern in einem Datenstrom direkt an den Indexierer weitergereicht.

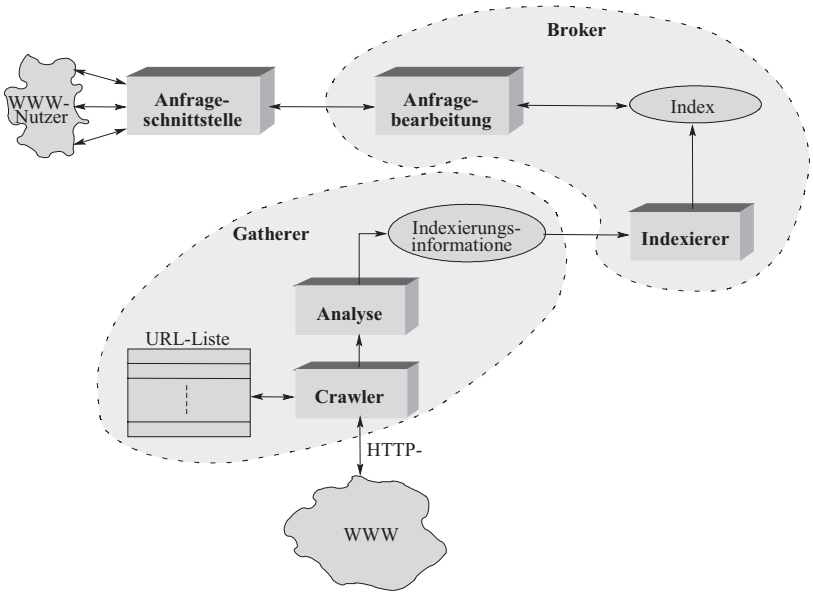


Abb. 9-2: *Zentralisierte Architektur*

Crawler sind Software-Agenten, die das WWW durchlaufen und neue bzw. geänderte Seiten an den Indexierer senden. Ein Crawler »kriecht« nicht, wie sein Name vermuten lässt, von einem Server zum anderen, sondern er läuft im lokalen System und sendet Anfragen an die entfernten Web-Server. Die Links, die auf einer angeforderten Seite enthalten sind, können weiterverfolgt werden, so dass schrittweise der Suchraum des Crawlers erweitert wird. Die aus der Analyse gewonnenen Metadaten werden anschließend indexiert. Der Index dient als Basis zur Beantwortung der Anfragen, die von verschiedenen Clients über das Web an die Suchmaschine gestellt werden. In der Architektur sind die einzelnen Bestandteile zu zwei logischen Komponenten zusammengefasst, die mit der Suchmaschine *Harvest* eingeführt wurden: dem *Gatherer*, der für das Sammeln und Extrahieren von Indexierungsinformationen zuständig ist, und dem *Broker*, der einen Indexierungsmechanismus und eine Anfrageschnittstelle auf die gesammelten Daten bereitstellt.

Als Indexierungsinformationen werden alle Daten über ein angefordertes Dokument bezeichnet, die im Index abgelegt werden und somit durchsuchbar sind.

Sie können sowohl Metadaten (Informationen über das Dokument wie Schlüsselwörter und Abstrakt) als auch den Volltext (Dokumentinhalt ohne Formatierungen) enthalten. Der Umfang der abgespeicherten Informationen variiert von Suchmaschine zu Suchmaschine – standardmäßig wird der Volltext indexiert, es gibt aber auch Ausnahmen wie Lycos, die nur einen Teil des Textes verwenden.

Der Nachteil dieser Architektur liegt in der Erzeugung unnötiger Netzlast durch die direkte Zuordnung eines Crawlers zu einem Indexierer, d.h., es wird immer ein kontinuierlicher Datenstrom zwischen Crawler und Indexierer aufrechterhalten. Dadurch ergeben sich folgende Probleme:

- Ein Crawler kann nicht lokal auf einem einzusammelnden Web-Server installiert werden (in diesem Fall könnten nur geänderte bzw. neue Dokumente an den Indexierer geschickt werden). Alle Dokumente müssen somit über das Web angefordert werden, unabhängig davon, ob sie geändert wurden oder nicht.
- Wenn sich die von verschiedenen Suchmaschinen eingesammelten Web-Bereiche überlappen, dann werden Dokumente mehrfach eingesammelt. Man stelle sich zum Beispiel eine Firma mit mehreren Abteilungen vor, die jeweils eigene Web-Präsentationen haben. Ziel ist es, nicht nur über dem Gesamtdatenbestand der Firma, sondern auch in den Informationen der einzelnen Abteilungen separat zu suchen. Die Web-Server der einzelnen Abteilungen müssen dann sowohl von der Abteilungssuchmaschine als auch von der Hauptsuchmaschine der Firma abgesammelt werden, da ein Crawler nur Informationen an einen Indexierer geben kann.

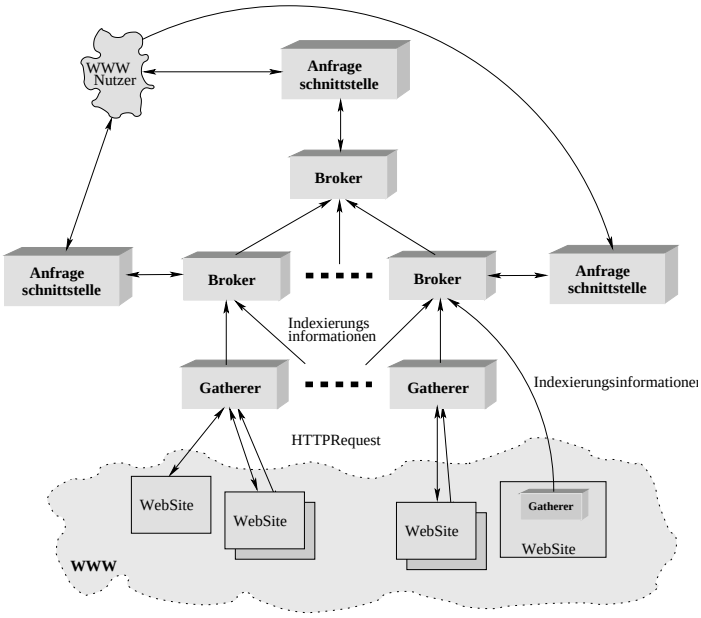


Abb. 9-3: *Verteilte Architektur*

Da das Anfordern der Dokumente von den verschiedenen Web-Servern sehr zeitaufwändig ist, kann ein Crawler in einem bestimmten Zeitraum nur einen sehr kleinen Teil des WWW einsammeln. Moderne Suchmaschinen mit zentralisierter Architektur setzen deshalb parallel laufende Crawler ein, die einem Indexierer zugeordnet sind[2] .

Die genannten Probleme können mit einer verteilten Architektur (siehe Abbildung 9-3) gelöst werden, bei der eine Zwischenspeicherung der gesammelten Informationen im Gatherer vorgenommen wird [BaRi99], so dass eine kontinuierliche Verbindung zwischen beiden Komponenten nicht mehr erforderlich ist.

Die Kommunikation zwischen den Komponenten erfolgt über ein festgelegtes Protokoll. Der Einsammelprozess kann durch die Verteilung auf mehrere Gatherer effizienter gestaltet werden. Der Nachteil dieser Variante ist die notwendige Koordinierung der Web-Bereiche, die von den verschiedenen Gatherern eingesammelt werden sollen[3]. Gatherer und Broker können wie folgt verteilt und miteinander kombiniert werden:

- Der Gatherer kann lokal auf dem zu indexierenden Web-Server laufen, wodurch die Netzlast erheblich reduziert wird – zum einen ist ein Zugriff über das Dateisystem möglich und zum anderen werden nur neue bzw. geänderte Dokumentbeschreibungen (keine vollständigen Dokumente) an den Broker übertragen.
- Ein Broker kann Informationen von einem oder mehreren Gatherern anfordern und somit einen Gesamtindex über verschiedene Web-Bereiche bilden.
- Ein Gatherer kann seine Informationen an mehrere Broker verteilen. Dadurch kann ein nochmaliges Einsammeln vermieden werden.
- Broker können wiederum Informationen von anderen Brokern anfordern.

Diese Kombinationsmöglichkeiten gestatten eine stufenförmige Anordnung der durchsuchbaren Informationen. Als Lösung für das bei der zentralisierten Architektur vorgestellte Firmenbeispiel bietet sich folgende Aufteilung an: Pro Abteilung wird ein Broker aufgesetzt, der von einem oder mehreren Gatherern mit den gesammelten Web-Informationen der Abteilung gespeist wird. Alle Abteilungsbroker wiederum geben ihre Informationen an den Hauptbroker weiter, der eine Suche über dem Gesamtdatenbestand ermöglicht. Eine Verringerung der Netzlast kann erreicht werden, indem die Gatherer auf den Web-Servern der Abteilungen installiert werden, da dann das Einsammeln lokal erfolgen kann.

Für die Kommunikation zwischen den Brokern gibt es zwei Möglichkeiten: Entweder werden die Indexierungsinformationen weitergereicht (1), oder der übergeordnete Broker reicht die Anfrage an den untergeordneten Broker weiter (2). Im ersten Fall werden die Indexierungsinformationen dupliziert und im Index des übergeordneten Brokers abgelegt (speicheraufwändig). Bei der zweiten Reali-

2. Die Verbindung eines Crawler zu mehreren Indexierern ist weiterhin nicht möglich.
3. Das gleiche Problem tritt auch bei parallel laufenden Crawlern in der zentralisierten Architektur auf.

sierung wird die Anfrage vom untergeordneten Broker beantwortet. Das Ergebnis muss dann im übergeordneten Broker in einem weiteren Schritt mit dem Resultat der Anfrage über dem eigenen Index bzw. mit den Resultaten weiterer Broker gemischt werden (zeitaufwändig).

9.3.3 Gatherer

Ein Gatherer startet den Sammelprozess mit einer Menge gegebener URLs. Dabei handelt es sich im Allgemeinen um viel besuchte Seiten. Manche Suchmaschinen bieten den Nutzern auch die Möglichkeit, URLs zu der Startmenge hinzuzufügen. Im ersten Schritt werden die Dokumente, die unter diesen URLs zu finden sind, angefordert. Handelt es sich bei dem angeforderten Dokument um eine HTML-Seite, dann können die darin enthaltenen Links extrahiert und weiterverfolgt werden.

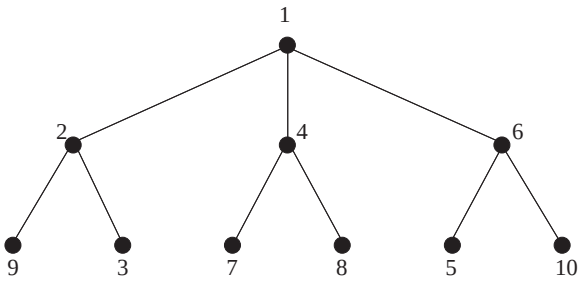


Abb. 9-4: *Beispiel für einen Linkgraphen*

Das Verfahren der Tiefensuche dagegen folgt zunächst jedem Link, der auf ein anderes Dokument verweist, bevor die benachbarten Knoten durchsucht werden. Ein Tiefendurchlauf durch den Beispielgraphen ergibt ausgehend von Knoten 1 die Reihenfolge {2, 9, 3}, {4, 7, 8} und {6, 5, 10}. Gatherer verwenden häufig die Breitensuche, da viele Web-Server hierarchisch organisiert sind und der Gatherer so schneller eine thematisch breitere Menge an Dokumenten finden kann. Eine Tiefensuche findet dagegen schneller individuelle Homepages mit Verweisen zu anderen neuen Servern und damit neuen Seiten.

Die Größe des Linkgraphen kann im Allgemeinen durch *URL-* bzw. *Host-Filter* begrenzt werden. Die Definition entsprechender Filter erlaubt das Ausschließen bestimmter Pfade, Dateiformate und Web-Server von der Indexierung. Weitere Einschränkungsmöglichkeiten liegen in der Angabe der maximalen Suchtiefe eines Server und der Maximalanzahl von Dokumenten, die von einem Server indexiert werden sollen. Wird die Maximalanzahl der einzusammelnden Dokumente begrenzt, dann hat das angewandte Suchverfahren entscheidende Auswirkungen auf die Art der eingesammelten Informationen. Bei einer Breitensuche wird eine breite, aber flache Abdeckung des Themenspektrums eines Web-Servers erreicht, während die Tiefensuche eine tiefe, aber schmale Abdeckung bedingt.

Jedes gesammelte Dokument wird an die Analysekomponente weitergereicht, die für dessen Verarbeitung und die Extraktion der zu indexierenden Informationen zuständig ist. Die meisten Suchmaschinen können nicht nur HTML-Seiten, sondern auch Dokumente im Postscript-Format oder PDF indexieren. Ermöglicht wird dies durch die Anwendung verschiedener Extraktionstechniken in Abhängigkeit vom aktuellen Dokumentformat. Die extrahierten Informationen werden dann entweder direkt an den Indexierer weitergereicht (zentralisierte Architektur) oder zunächst in einer eigenen Datenbasis gespeichert (verteilte Architektur), eine Sicherung der eigentlichen Dokumente erfolgt im Allgemeinen nicht. Dadurch kommt es bei der Suche häufig vor, dass als Treffer angezeigte Links auf nicht mehr vorhandene Webseiten zeigen oder aber die Seiten bereits geändert wurden und die gewünschten Informationen nicht mehr enthalten. Aus diesem Grund muss der Zeitraum, in dem eine Website neu besucht wird, möglichst gering gewählt werden, damit weitestgehend aktuelle Inhalte als Basis für die Suche dienen.

9.3.4 Broker mit Indexierer

Der Broker indexiert die vom Gatherer gelieferten Daten. Bei der verteilten Architektur wird dazu eine zusätzliche Collector-Komponente benötigt, die alle Daten von den eingestellten Gatherern bzw. untergeordneten Brokern anfordert. Anfragen an den Broker können über die Anfragebearbeitungskomponente gestellt werden, die zur Beantwortung auf den erstellten Index zurückgreift.

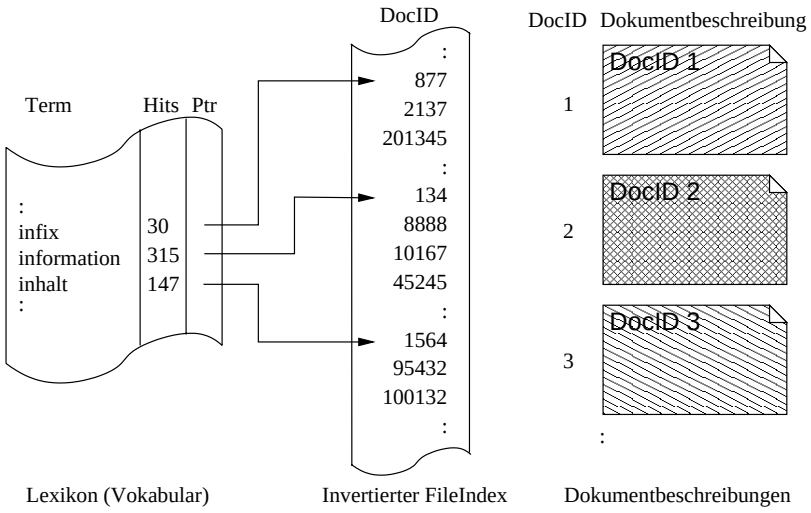


Abb. 9-5: *Aufbau von invertierten Listen in Suchmaschinen*

In der Abbildung werden die Zeiger auf Dokumente mit Hilfe von Dokument-Identifikatoren (`DocID`) realisiert, die Zuordnung von Term und Dokumentenliste

erfolgt über den Verweis `Ptr`. Das Lexikon kann neben den eigentlichen Termen auch die Häufigkeiten der Terme in der Dokumentensammlung (`Hits`) aufnehmen. Zur Realisierung einer effizienten Termsuche wird das Lexikon üblicherweise als sortierte Liste, B-Baum oder mittels Hashing-Verfahren organisiert. Vervollständigt wird der Index durch eine Beschreibung der Webseite, die in der Ergebnispräsentation für eine Kurzdarstellung der Seite nutzbar ist und Angaben wie URL, Erzeugungsdatum, Titel und Kurzfassung umfasst. Der Inhalt der Kurzfassung einer Seite variiert bei den einzelnen Suchmaschinen – einige nutzen die ersten Zeilen des Dokumentes, andere verwenden z.B. Abschnittsüberschriften. Im Extremfall wird der gesamte Volltext des Dokumentes gespeichert, um beispielsweise eine Phrasen- bzw. Proximity-Suche zu ermöglichen.

Zur Beantwortung einer Anfrage wird zunächst eine Suche im Vokabular der invertierten Liste vorgenommen. Wenn der gesuchte Term in der invertierten Liste enthalten ist, dann kommen alle zugeordneten Dokumente als Treffer in Frage. Bei kombinierten Anfragen erfolgt am Ende eine Zusammenfassung der Einzelergebnisse zu einem Gesamtergebnis. Die einem Term zugeordnete Dokumentenliste ist gewöhnlich in aufsteigender Reihenfolge der Dokument-Identifikatoren sortiert, so dass Verfahren zum Mischen der Listen eingesetzt werden können, deren Zeitbedarf linear zur Listenlänge ist.

Die in der invertierten Liste zu jedem Term abgelegten Informationen können mehr oder weniger detailliert sein und beeinflussen damit den Speicheraufwand wie auch die Anfragemöglichkeiten. Bei der in Abbildung 9.1 vorgestellten Variante handelt es sich um einen mittelfeinen Index, d.h., zu jedem Term werden die Dokumente festgehalten, in dem sie vorkommen. Ein feiner Index dagegen nimmt zusätzlich die Position des Terms innerhalb des Dokuments auf. Dies ist zwar sehr speicheraufwändig, erlaubt auf der anderen Seite aber Anfragen mit Phrasen und »Entfernungsangaben« (Proximity-Suche) ohne ein nachträgliches Durchsuchen der in Frage kommenden Dokumente. Der Speicherbedarf kann mit einem groben Index stark reduziert werden, d.h., hier werden nur Zeiger auf logische Blöcke verwendet, die mehrere Dokumente zusammenfassen. Dadurch kann sowohl eine Reduzierung der Zeigergröße als auch eine Reduzierung der Anzahl der Zeiger erreicht werden. Bei Anfragen zeigt sich aber der Nachteil dieser Variante, da in allen über die invertierte Liste ermittelten Blöcken sequenziell nach den Ergebnisdokumenten gesucht werden muss.

Die Größe eines unkomprimierten Indexes liegt in Abhängigkeit von der Granularität zwischen 10 bis 100% des indexierten Datenbestandes. Durch die Eliminierung von Stoppwörtern kann diese Größe signifikant verringert werden, da es sich hierbei um Terme handelt, die in sehr vielen Dokumenten vorkommen. Ein Ignorieren der Stoppwörter erfolgt aber nicht nur bei der Indexierung, sondern auch innerhalb einer Anfrage.

Invertierte Listen können nicht nur als Zugriffsstruktur für boolesches Retrieval verwendet werden. Sie eignen sich ebenfalls für ein Retrieval mit Hilfe des Vektorraummodells (siehe Abschnitt 9.2.2). In diesem Modell dient der Winkel zwischen Anfrage- und Dokumentvektor als Maß für die Ähnlichkeit. Zur

Berechnung der Ähnlichkeit kann der Kosinus des Winkels bestimmt werden, wenn für jeden Term T_m

- im Lexikon die Anzahl der Dokumente d_m, in denen T_m innerhalb der Dokumentensammlung auftritt, und
- in der invertierten Liste die Häufigkeit f_{mn} innerhalb des Dokuments D_n

abgespeichert wird. Bewertet werden alle Dokumente aus der Vereinigungsmenge der invertierten Listen für alle Anfrageterme. Somit können auch Dokumente im Ergebnis vorhanden sein, die nicht alle Anfrageterme enthalten. Die Berechnung der Ähnlichkeit erfolgt schrittweise bei der Bestimmung des Anfrageergebnisses – nacheinander wird für jeden Term T_m der Anfrage Q die invertierte Liste gelesen. Während des Lesens wird das Termgewicht w_{mn} für ein Dokument D_n mit einer in Abschnitt 9.2.3 vorgestellten Formel ermittelt und mit dem Termgewicht w_{mq} in der Anfrage multipliziert. Die Ergebnisse, die sich pro Anfrageterm T_q für ein Dokument D_n ergeben, werden anschließend aufsummiert. Nach der Verarbeitung aller Anfrageterme liegt dann für jedes im Anfrageergebnis vorkommende Dokument eine Endsumme vor, die nach der Normierung dem Kosinus zwischen Anfrage- und Dokumentvektor entspricht. Im letzten Schritt werden die gefundenen Dokumente gemäß ihrer Relevanz absteigend sortiert.

Die booleschen Operationen `AND` und `OR` lassen sich mit dem Vektorraummodell ebenfalls realisieren, wenn die Gewichte der einzelnen Terme mit Null (Term kommt nicht vor) oder Eins (Term kommt mindestens einmal vor) festgelegt werden. Das Ergebnis liefert in diesem Fall die Anzahl der in einem Dokument enthaltenen Anfrageterme und muss bei der `AND`-Operation mit der Anzahl der Suchterme übereinstimmen.

9.3.5 Ranking

Im Gegensatz zu den bisher in Abschnitt 9.2.3 vorgestellten Ranking-Algorithmen im Information Retrieval werden nun erweiterte sowie auf Internet-Suchmaschinen eingeschränkte Konzepte vorgestellt.

Das Ranking ist ein zweistufiger Prozess innerhalb der Suchmaschine. Er umfasst eine Analysephase im Gathering und eine Berechnungsphase bei der Anfrageverarbeitung im Broker. Das Ranking basiert auf verschiedenen Annahmen bzw. Heuristiken, die zum Teil von den ursprünglichen IR-Systemen übernommen wurden (siehe auch Abschnitt 9.2.3). Die konkrete Anwendung dieser Heuristiken im WWW bietet eine Vielzahl von Anpassungs- und Verbesserungsmöglichkeiten.

Aufgrund großer Ergebnismengen bei oft gestellten, einfachen Anfragen ist ein Ranking bei Internet-Suchmaschinen unerlässlich. Die Schwierigkeiten im WWW liegen in der Heterogenität der Dokumente (das WWW hält eine Vielzahl von Dokumenttypen vor) und der unterschiedlichen Granularität bei der Aufbereitung der Informationen innerhalb der Dokumente (z.B. gibt es zu einem Thema neben Dokumenten mit zwei Sätzen auch ganze Buchmanuskripte). Im Gegensatz zu

herkömmlichen IR-Systemen sind zusätzliche Informationen durch interne und externe Strukturinformationen vorhanden und auswertbar. In HTML-Dokumenten sind beispielweise so genannte Tags als Strukturbeschreibung zu finden, und als externe Struktur werden Links zwischen Dokumenten definiert.

Spezielle Ranking-Heuristiken im WWW

Zu den IR-Heuristiken werden weitere speziell an die Gegebenheiten des Web angepasste Ranking-Heuristiken vorgestellt.

- *Metadaten:* Durch explizite Auszeichnung von Metadaten, also zusätzlichen Daten zum Dokument wie Autor, Keywords etc., sind weitere Möglichkeiten der Anfrage und des Rankings gegeben. Einerseits wird dadurch der Index erweitert, andererseits kann die Anfrage innerhalb der Nutzerschnittstelle attributiert werden, so dass ein Nutzer bspw. explizit nach dem Autor »Meier« suchen kann. Das Ranking kann nun so aufgebaut werden, dass Suchtreffer, deren Terme in einem Attributwert gefunden wurden, höher gerankt werden (z.B. bei einer Suche nach »Meier AND IR«).
- *Dokumentspezifische Strukturdaten:* Eine weitere Möglichkeit des Ranking-Einflusses besteht durch Strukturdaten. Werden Terme in einer logisch wichtigen Strukturumgebung gefunden, können die zugehörigen Dokumente im Ranking[4] aufgewertet werden. Zum Beispiel können Dokumente mit im Titel enthaltenen Anfragetermen höher bewertet werden.
- *Dokumentübergreifende Strukturdaten:* Dokumentübergreifende Informationen können aus den Linkstrukturen und der Dokumenthierarchie innerhalb eines Web-Servers gewonnen werden. Die Hierarchiestufe eines Dokuments spiegelt sich in der Länge des in der zugehörigen URL enthaltenen Pfades wider. Zu den Linkstrukturen gibt es im Wesentlichen zwei Algorithmen: das HITS-Verfahren von Kleinberg [Klei98] und der PageRank-Algorithmus von [Page98]. Beide Verfahren bewerten die Popularität von Dokumenten anhand der Link-Beziehungen. Dabei werden ein- und ausgehende Links im gerichteten Dokument-Link-Graph gegeneinander gewertet. PageRank basiert auf der Übernahme von Ranking-Werten aus Dokumenten, die einen Link auf das zu berechnende Dokument haben, wobei die Werte auf alle verwiesenen Dokumente aufgeteilt werden. Die Summe aus den Werten aller eingehender Links bildet dann das Ranking-Gewicht. Beim HITS-Algorithmus zählen zur Bewertung nur Links von Dokumenten, die relevant zur Anfrage sind. Dadurch können auch Dokumente einen hohen Ranking-Wert bekommen, die ansonsten aufgrund nicht vorhandener Anfrageterme einen hohen Relevanzabstand zur Anfrage haben.
- *Temporäre Aspekte:* Zeitliche Aspekte spielen im WWW eine gehobene Rolle, da ein Großteil der Dokumente sehr schnelllebig ist. Als Metadaten kodierte

4. Strukturheuristiken wie die Titel-Heuristik

Erstellungs- bzw. Update-Zeiten oder einfach der Einsammelzeitpunkt ergeben einen möglichen Modifizierer für die Ranking-Werte.

Kombination der Ranking-Konzepte

Die verschiedenen Konzepte müssen nun in einer Ranking-Funktion zusammengefasst werden.

Das Zusammenführen von verschiedenen Ranking-Konzepten in eine Ranking-Funktion geschieht auf unterschiedlichen Ebenen. Einige Konzepte bilden die Basisgewichte für das jeweilige Dokument, andere Konzepte sind eher als Modifikator von einzelnen Gewichten zu sehen und wiederum andere modifizieren den gesamten Rankingwert.

```
Anfrage: "Datenbanksysteme AND Heuer'' => TV=(1.0,1.0)

Dokument1: TV=(0.7,0.0) => 0.71; Link=0.9; Date=1.0
Dokument2: TV=(0.1,0.9) => 0.78; Link=0.5; Date=0.9
Dokument3: TV=(0.8,0.2) => 0.85; Link=0.6; Date=0.8
Dokument4: TV=(0.6,0.8) => 0.99; Link=0.3; Date=0.9

Heuristikgewichte: W(TV)=0.5; W(Link)=0.2; W(Date)=0.3

Dok1: 0.833 => 1.
Dok2: 0.76 => 4.
Dok3: 0.78 => 3.
Dok4: 0.825 => 2.
```

Beispiel 9-1: *Vereinfachtes Beispiel eines Rankings mit drei verschiedenen Heuristiken*

Das Beispiel 9-1 zeigt an einer einfachen Anfrage und drei voneinander unabhängigen Ranking-Heuristiken, wie ein Ranking im WWW aussehen kann. Die drei Heuristiken basieren auf Termvektoren (`TV`), Linkstrukturen (`Link` – z.B. nach PageRank) und Datumsangaben (`Date` – z.B. der Abstand zwischen der letzten Änderung im Dokument und dem Anfragezeitpunkt). Der Rankingwert der Termvektoren wurde mittels Kosinus-Abstandsmaß zum Anfragevektor berechnet. Die Kombination der drei Heuristiken erfolgt mit unterschiedlichen Gewichten. Interessanterweise ist die abschließende Ranking-Reihenfolge mit keiner Reihenfolge für die einzelnen Heuristiken identisch.

Ein automatischer Ansatz zur Kombination von Ranking-Funktionen wird von [Rape01] vorgestellt. Dabei wird eine so genannte Multi-Heuristic-Ranking-Funktion definiert, die verschiedene Heuristiken mit unterschiedlichen Ranking-Werten kombiniert. Die einzelnen Heuristiken werden mittels Gewichten auf eine Initialisierungsmenge von möglichen Anfragen angepasst.

Ranking in verteilten Suchmaschinen

Eine Kombination von Ranking-Funktionalitäten ist bei verteilten Suchmaschinen und Meta-Suchmaschinen unerlässlich. Bei heterogen verteilten Suchmaschi-

nen mit verteilten Indexen und somit möglicherweise verschiedenen Ranking-Konzepten sowie bei heterogenen Meta-Suchmaschinen bestehen zwei wesentliche Probleme. Erstens werden verschiedene Ranking-Konzepte unterstützt, und zweitens werden die Ranking-Werte unterschiedlich kombiniert und normiert.

Die optimale Lösung erreicht man, wenn die verteilten Suchmaschinen kooperieren. Dann kann ein globales Re-Ranking mit genau bekannten Gewichten und Heuristiken der einzelnen Suchmaschinen berechnet werden. Existiert keine Kooperation, sind also keine genauen Daten zu den lokalen Ranking-Funktionen vorhanden, muss das globale Ranking abgeschätzt werden. [Call00] hat dafür mehrere Heuristiken zusammengefasst. Eine einfache Heuristik besteht darin, die einzelnen Suchressourcen mit einer Bewertung zu belegen und somit eine gewisse Abstufung zu erreichen. Die berechneten Ranking-Werte hängen vom Ursprungswert, von der Anzahl der Suchressourcen und von der Bewertung ab.

9.3.6 Anbindung von Datenbanken an Suchmaschinen

Die Gatherer der aktuell verfügbaren Suchmaschinen sammeln im Allgemeinen nur Daten ein, die im so genannten öffentlich indexierbaren Web vorhanden sind. Informationen aus Web-Datenbanken, die etwa 80% der Web-Inhalte ausmachen [LaGi98], werden somit ignoriert und deshalb oft als »Hidden Web« bezeichnet. Solche Datenbanken haben typischerweise eine Web-Anfrageschnittstelle, die ein HTML-Formular enthält. Der Zugriff auf die Datenbank ist dann über dynamisch generierte Seiten möglich, die als Antwort auf die Nutzeranfrage geliefert werden und eine implizit vorliegende, aber nicht sichtbare Struktur haben.

Es gibt verschiedene Varianten für die Integration von Datenbanken in Suchmaschinen, die sich in der Zugriffsmöglichkeit des Gatherer auf die Datenbank und in den bereitgestellten Informationen unterscheiden:

- *Kooperative Datenbankanbieter* ermöglichen den strukturierten Zugriff auf den Datenbankinhalt über JDBC oder ähnliche Mechanismen. In diesem Fall kann die volle Funktionalität einer Anfragesprache in der Datenbankanbindung ausgenutzt werden.
- Bei *nicht-kooperativen Anbietern* können Anfragen nur über ein WWW-Formular gestellt werden. Das Schema der Datenbank sowie nutzbare Anfragefunktionen sind nicht bekannt bzw. können nicht an die darunter liegende Datenbank übergeben werden.
- Im Falle von *semikooperativen Anbietern* ist der Zugriff ebenfalls nur über die Web-Schnittstelle möglich, die Funktionalität des Anfrageformulars und die Struktur der Ergebnisseiten wird aber über spezielle, vom Datenbankanbieter bereitgestellte Metadaten spezifiziert.

Der kooperative Ansatz wurde z.B. in der AltaVista Search Engine (AVSE, Details siehe Abschnitt 9.4) umgesetzt. Zur Indexierung von relationalen Datenbanken steht hier ein eigener Collector zur Verfügung [AVAH00], der auf einen entfernten

Datenbank-Server über JDBC zugreift. Nach dem Aufbau der Verbindung werden die Tupel unter Nutzung der Konfigurationsinformationen aus der Datenbank gelesen und anschließend indexiert. Jedes Tupel wird dabei als eigenes Dokument aufgefasst, dem eine URL mit dem gewählten Primärschlüssel als Parameter zugeordnet ist. Wenn ein Tupel im Anfrageergebnis enthalten ist, dann kann die AVSE über diese URL auf das korrespondierende Tupel in der Datenbank mit weiteren, nicht indexierten Attributen zugreifen. Der Nachteil dieser Realisierung liegt in der Notwendigkeit eines Direktzugriffs auf die Datenbank über das Web. Eine in der SWING-Suchmaschine umgesetzte Idee nutzt dagegen so genannte »lokale Summarizer«, die auf dem WWW-Server des Datenbankanbieters laufen [Webe01] – notwendige Datenbankzugriffe erfolgen hier lokal. Die Kommunikation zwischen Suchmaschine und Web-Datenbank kann somit über HTTP erfolgen und der Datenbankanbieter hat eine volle Kontrolle über die Daten, die indexiert werden. Die lokalen Summarizer stellen entsprechend der Konfigurationsangaben generierte Anfragen an die zu indexierende Datenbank und liefern den zu indexierenden Datenbankinhalt in einem Gesamtdokument an den Gatherer zurück.

Die Einbindung von Datenbanken ohne Kooperation erfordert ein automatisches Ausfüllen des Web-Formulars. Der an der Stanford Universität entwickelte Hidden Web Exposer (HiWE) ist ein aufgabenspezifischer Web Crawler, der solch einen Automatismus bietet [RaGa01]. Der HiWE greift beim Bestücken der Formulardaten auf eine endliche Menge von Konzepten einer Anwendungsdomäne zurück. Die Zuordnung eines Formularelements zu einem Konzept wird über textuelle Ähnlichkeiten zwischen der Beschreibung des Elements und der Konzeptbeschreibung vorgenommen. Jeder einem Konzept zugeordnete Wert besitzt ein Gewicht, das die Relevanz dieses Wertes für das Konzept beschreibt. Mit Hilfe dieser Gewichte können die »besten« Wertzuweisungen für die Formularelemente ermittelt werden. Die Ergebnisseiten werden wie normale Dokumente indexiert – eine inhaltliche Analyse zur Bestimmung einzelner Attributwerte wird nicht vorgenommen. Die flache Konzepthierarchie und die Nutzung von textuellen Ähnlichkeiten beschränkt die Nutzung auf sehr eingegrenzte Domänen.

Eine domänenunabhängige Indexierung über die Web-Schnittstelle ist im semikooperativen Fall mit Hilfe von Metadaten möglich [Webe01], in denen der Datenbankanbieter die Funktionalität des Anfrageformulars und den Aufbau der Ergebnisseiten beschreibt. Durch die Nutzung dieser Informationen können die Werte für die Formularelemente so gewählt werden, dass die Datenbank mit einer minimalen Zahl von Anfragen vollständig indexierbar ist. Die zu indexierenden Attribute können in den Ergebnisseiten durch die bereitgestellten Metadaten ebenfalls effizient bestimmt werden. Die Extraktion der Attributwerte spart zum einen Speicherplatz und erlaubt zum anderen eine attributierte Suche (bei entsprechender Unterstützung durch den Indexierer).

Die Tabelle 9-2 zeigt einen Vergleich der Ansätze zur Datenbankintegration. Im nicht-kooperativen Fall hat der Anbieter gar keinen Aufwand, um seine Datenbank indexierbar zu machen. Der kooperative Ansatz dagegen erfordert entweder die Installation eines lokalen Summarizers oder das Ermöglichen eines

Direktzugriffs auf die Datenbank mit anschließender Konfiguration des Indexierungsprozesses durch den Administrator der Suchmaschine.

Vergleichskriterium[a]	Ansatz		
	kooperativ	nicht kooperativ	semikooperativ
Integrationsaufwand	-	+	°
Ressourcenbelastung	+	-	°
Indexierungsdauer	++	-	°
Fehleranfälligkeit	+	-	°
Abdeckung Domäne	-	++	+
Abdeckung allgemein	-	-	+

a. Bewertung bezüglich des Vergleichskriteriums: - schlecht, ° durchschnittlich, + gut, ++ sehr gut

Tab. 9-2: *Vergleich der Ansätze zur Datenbankanbindung*

Die Ressourcenbelastung der Suchmaschine und des Web-Servers des Datenbankanbieters ergibt sich aus der Anzahl der erforderlichen Verbindungen und Datenbankanfragen. Bei voller Kooperation ist diese Belastung am geringsten, da hier nur ein HTTP-Request bzw. eine JDBC-Verbindung notwendig ist. Bei den anderen beiden Ansätzen ergibt sich die Anzahl der notwendigen Requests aus der Wahl der günstigsten Anfragebelegungen. Da die automatische Bestimmung der minimal notwendigen Anfragen kaum denkbar ist, sind im nicht-kooperativen Fall für eine vollständige Indexierung in der Regel mehr Anfragen erforderlich als beim semikooperativen Ansatz. Bei der Suchmaschine kommt beim Zugriff über die Web-Schnittstelle zusätzlich noch der Aufwand für die Analyse der Ergebnisseiten zum Erkennen möglicher Fehlerzustände bzw. zur Extrahierung der Attributwerte hinzu. Die Indexierungsdauer steigt aus den eben genannten Gründen mit abnehmendem Kooperationsumfang.

Die Fehleranfälligkeit[5] hängt bei voller Kooperation nur vom Datenbankschema ab. Da dieses sich in der Regel kaum ändert, ist die Fehleranfälligkeit gering. Beim nicht-kooperativen Ansatz ist die Fehleranfälligkeit durch die Mehrdeutigkeit der natürlichen Sprache hoch. Die im semikooperativen Fall eingesetzten Extraktionsregeln basieren auf HTML-Tags, die das Layout der Ergebnisseite bestimmen. Die Fehleranfälligkeit hängt hier von der Robustheit der definierten Regeln gegenüber Layoutänderungen ab.

Im nicht-kooperativen Fall kann innerhalb einer bestimmten Anwendungsdomäne eine vollständige Abdeckung[6] erreicht werden. Beim semikooperativen Ansatz sind dagegen nur Datenbanken indexierbar, die entsprechende Metadaten besitzen. Im kooperativen Fall ist nur die Einbindung von Datenbanken möglich, die

5. Der dauerhafte Zugriff auf den Web-Server bzw. die Datenbank wird hier vorausgesetzt.
6. Unter Abdeckung wird hier das Verhältnis der indexierbaren Datenbanken zu allen Datenbanken, deren Anfrageschnittstelle innerhalb eines Gatherer-Laufes eingesammelt wird, bezeichnet.

einen Zugriff erlauben und explizit bei der Suchmaschine angemeldet sind. Bei beliebigen Anwendungsdomänen ist dagegen der semikooperative Ansatz von Vorteil, da die Metadaten domänenunabhängig sind. Im nicht-kooperativen Fall können nur Datenbanken solcher Anwendungsdomänen indexiert werden, für die eine Wissensbeschreibung existiert.

9.4 Konkrete Suchmaschinen und Indexierer

Die meisten aktuell verfügbaren Suchmaschinen basieren auf einer zentralisierten Architektur; die hier vorgestellten Suchmaschinen Google und AltaVista gehören zu den meist genutzten. Als Vertreter für Suchmaschinen mit einer verteilten Architektur wird Harvest genauer vorgestellt.

Google

Die Suchmaschine Google (www.google.de) wurde ursprünglich von der Stanford University entwickelt. 1998 wurde das im Projekt entwickelte Ranking-Verfahren PageRank (s. Abschnitt 9.3.5) zum Patent angemeldet und das Projekt in eine eigenständige Firma überführt.

Google besitzt mit über 2 Milliarden Dokumenten momentan den größten Index unter den Suchmaschinen. Zu den besonderen Eigenschaften zählt vor allem ein sehr gutes Ranking mit Ausnutzung von Eigennamenerkennung, Wortabständen und dem PageRank-Algorithmus. Weiterhin bietet Google einen Web-Cache, der es ermöglicht, das tatsächlich indexierte Original auch bei Nichtvorhandensein der Webseite zu erhalten. Ein weiterer Vorteil des Caches ist die Möglichkeit, beim Zugriff auf die Dokumente die Suchterme hervorzuheben.

Es gibt bei Google die einfache und die erweiterte Suche. Die einfache Suche umfasst eine automatische UND-Suche, das Ausschließen von Termen und das Einschränken von URL-Domänen sowie eine Phrasensuche. Wildcards werden nicht unterstützt. Die erweiterte Suche bietet darüber hinaus eine ODER-Suche, bestimmte Einstellungen wie Sprache, Dateiformat, Datumsbegrenzung und fünf verschiedene Vorkommenspositionen innerhalb des Dokuments. Interessant ist auch die so genannte seitenspezifische Suche. Hier kann zum einen eine ähnliche Seite zu einer vorgegebenen gesucht werden und zum anderen ist die Suche nach Dokumenten möglich, die auf eine gegebene Seite referenzieren.

Google bietet auch eine sehr einfache Art der Personalisierung über Cookies, mit denen Sprach- und Ergebnislayout-Einstellungen vorgenommen werden können.

Zu den technischen Details von Google kann momentan nur auf die ursprünglichen Publikationen an der Stanford University zurückgegriffen werden. Die Google-Architektur besteht nach [Page98] aus einer Menge von Crawlern, die von einem URL-Server zu durchsuchende URL-Listen erhalten. Ein Store-Server nimmt die gefundenen Seiten entgegen, komprimiert sie und sammelt sie in einem Repository. Der Indexer nimmt die Dokumente heraus, dekomprimiert und ana-

lysiert sie. Die extrahierten Link-Referenzen werden mit zusätzlichen Umgebungsdaten in eine spezielle Link-Datenbank abgelegt und die neuen URLs an den URL-Server übergeben. Außerdem werden Link-Paare von URLs bzw. Dokument-Identifiern gebildet, die für den PageRank-Algorithmus benötigt werden. Der eigentliche Index besteht aus den Termhäufigkeiten pro Dokument, den Termpositionen und einigen Layoutdaten wie Fonts. Der Index wird auf mehrere so genannte Barrels verteilt. Ein Sortierer ordnet die Barrels und bildet invertierte Listen. Eine weitere Liste sowie das Lexikon mit Wortidentifikatoren und einer Referenz auf den Offset in der invertierten Liste wird dem Suchwerkzeug übergeben. Das Suchwerkzeug ist an einen Web-Server gebunden und benötigt das Lexikon, die invertierten Listen und die PageRank-Werte zur Beantwortung der Anfragen.

1998 hatte Google 24 Mio. Dokumente mit 259 Mio. Links gesammelt. Dies entsprach einem Repository-Umfang von knapp 148 GB (komprimiert 54 GB). Das Lexikon umfasste 14 Mio. Wörter und wurde so optimiert, dass es in den Hauptspeicher von damals 256 MB passte. Damit Google bei einer solchen Masse an Daten einen effizienten Zugriff bieten kann, wurden Index und Lexikon bis auf Bit-Ebene genau optimiert. Außerdem ist das System auch dahingehend optimiert, Festplattenzugriffe zu minimieren.

AltaVista

Die AltaVista Search Engine (AVSE) [AVAH00] ist eine modifizierte Version der Software, die zum Betrieb der bekannten WWW-Suchmaschine genutzt wird. Diese Suchmaschine ist ebenfalls ein Vertreter der zentralisierten Architektur und kann nicht nur Webseiten, sondern auch File-Server und Datenbanken indexieren. Für jede dieser Datenquellen existiert ein spezialisierter Crawler, der hier als *Collector* bezeichnet wird. Die Suchmaschine bietet die Erstellung mehrerer Indexe, wobei ein Index Daten von mehreren Collectoren erhalten kann. Nutzeranfragen werden immer über die Indexe beantwortet, die dafür spezifiziert wurden.

Die Indexierung erfolgt auf Wortbasis, der Speicherbedarf eines Dokuments im Index beträgt etwa 30% des Orginaltextes. Mit Hilfe der Wortposition können erweiterte Anfragemöglichkeiten wie Phrasen- und Proximity-Suche oder auch die Suche von Wörtern innerhalb eines Feldes realisiert werden, bei denen die Beziehung der Wortpositionen zueinander von Bedeutung ist. Wörter, die Großbuchstaben enthalten, werden zweimal indexiert – einmal in der Originalschreibweise und einmal in Kleinbuchstaben. Beide Versionen haben die gleiche Position, so dass das Wort bei Anfragen mit und ohne Beachtung von Groß- und Kleinschreibung gefunden werden kann. Gebräuchliche HTML-Meta-Tags werden standardmäßig indexiert, weitere können je nach Anwendungsfall dazukommen. Die indexierten Meta-Tags können dann in einer strukturierten Suche wie `keywords: database` verwendet werden. Der Index kann gleichzeitig verändert und angefragt werden, so dass jederzeit neue Dokumente hinzugefügt, existierende geändert und alte gelöscht werden können. Da sehr viele Update-Operationen zu langen Antwortzeiten führen, können auch zwei Indexe verwendet werden – einer für Anfragen und einer für die Durchführung der Updates.

Das Search Developer's Kit (SDK) [AVDK00] erlaubt anwendungsspezifische Anpassungen der AVSE. Für die Dokumente, die in den Index aufgenommen werden sollen, können Kategorien definiert werden. Dadurch ist es möglich, die Ergebnismenge einer Anfrage auf die spezifizierten Kategorien einzuschränken. Weiterhin können eigene Ranking-Werte für die Ergebnissortierung deklariert werden. Soll z.B. in einer E-Commerce-Anwendung die Sortierung von Angeboten nach dem Preis möglich sein, dann muss einfach nur ein neuer Wertetyp namens `Price`[7] definiert werden, nach dem die Sortierung erfolgt.

Die AVSE ermöglicht einfache Textsuche, boolesches Retrieval und Anfragen auf Basis des Vektorraummodells. Das Ranking basiert auf den Gewichtswerten der einzelnen Suchwörter und einer Relevanzbeurteilung jedes Dokuments, das die Suchanfrage erfüllt. Das Gewicht eines Wortes wird durch seine Auftrittshäufigkeit im gesamten Index bestimmt. Ein weniger häufig auftretendes Wort wird dabei als relevanter erachtet und erhält ein höheres Gewicht als ein Wort, das sehr häufig auftritt. Die Relevanzbeurteilung eines Dokumentes erfolgt nun unter Berücksichtigung der Anzahl der enthaltenen Suchwörter und ihrer dazugehörigen Gewichte. Das Dokument mit den meisten übereinstimmenden Suchwörtern und den höchsten Gewichtswerten weist somit die höchste Relevanz auf. Die Position des Wortes und die Häufigkeit seines Auftretens innerhalb eines einzelnen Dokumentes hat keinen Einfluss auf das Ranking.

Harvest

Harvest [HSWL01], ursprünglich von der University of Colorado entwickelt, ist eine der ersten Suchmaschinen mit einer verteilten Architektur und nach wie vor mit sehr vielen Installationen im WWW vorhanden. Die Verteilung geschieht zwischen Gatherer und Broker sowie zwischen verschiedenen Brokern. Es können mit beliebigen Hierarchieebenen beliebig große Gatherer/Broker- bzw. Broker/Broker-Netze aufgebaut werden.

Der *Gatherer* in Harvest besteht im Wesentlichen aus drei Komponenten. Die erste Komponente bereitet das eigentliche Sammeln vor, indem veraltete Dokumente aus dem lokalen Repository gelöscht und die zu aktualisierenden in einem Refresh-Repository vorgehalten werden. Das Essence-System des Gatherer arbeitet zuerst die Liste der zu erneuernden und dann die neu zu durchsuchendend Web-Dokumente und Web-Domänen ab. Zu erneuernde Dokumente werden anhand einer Prüfsumme auf Veränderungen kontrolliert. Unveränderte Dokumente müssen die weiteren Analyseschritte nicht durchlaufen. Zu jedem gesammelten Dokument wird zunächst der Typ ermittelt. Danach wird die Seite durch den zugeordneten Summarizer verarbeitet. Dieser parst das Dokument und liefert den Volltext sowie Metadaten in strukturierter Form zurück, die in *Harvest* durch das SOIF[8] definiert ist. Die gewonnenen Daten werden in lokalen Repositories für neue und aktualisierte Dokumente gespeichert. Die dritte Komponente fasst die

7. Bei der Definition wird der kleinst- und der größtmögliche Wert benötigt.
8. SOIF – Summary Object Interchange Format: Menge von Attribut/Wert-Paaren

ermittelten Daten aus den beiden generierten und dem ursprünglichen Dokument-Repository zu einem aktuellen Dokument-Repository zusammen.

Der *Broker* kann nun mit seinem Collector auf die Daten des Gatherer zugreifen. Der Broker indexiert diese Daten und stellt mit dem Query-Manager eine Zugriffsmöglichkeit bereit. Die Indexierer–Software wird extern eingebunden und kann einfach ausgetauscht werden. Ein Nutzer kann seine Suchanfrage über eine Anfrageschnittstelle an das System stellen. Der Broker sucht in seinen Datenbeständen die gewünschten Informationen heraus und präsentiert sie in einer bewerteten URL-Trefferliste. Die Anfragefunktionalität hängt vom eingebundenen Indexierer ab. Dazu muss die Anfrageschnittstelle zusätzlich angepasst werden.

Harvest ist relativ modular aufgebaut, so dass neben direkten Schnittstellen zur Erweiterung auch einzelne Module, wie Summarizer, Indexierer oder die Suchschnittstelle austauschbar sind.

mg

Der Schwerpunkt des Volltext-Retrieval-Systems mg (Managing Gigabytes) [WiMB99] liegt, dem Namen entsprechend, in der Behandlung sehr großer Datenmengen. Das mg-System ist nur für statische Dokumentsammlungen geeignet, da momentan das Einfügen neuer Dokumente ohne eine erneute Verarbeitung aller Dokumente nicht möglich ist. Neben Text-Dokumenten können auch Bilder verwaltet werden.

Um Speicherplatz zu sparen, werden sowohl die Dokumente als auch der Index komprimiert – zur Textkomprimierung wird beispielsweise der Hofmann-Kodierer genutzt, der eine gute Kompressionsrate und eine schnelle Dekompression bietet. Mit Hilfe der verschiedenen Kompressionsverfahren kann der Speicherbedarf der Datenbasis auf 25% und die Indexgröße auf 10% bezüglich der Originaldaten reduziert werden.

Als Index wird eine invertierte Liste auf Wortbasis mit Abspeicherung der Worthäufigkeit innerhalb des Dokuments und innerhalb der Datenbasis genutzt. Somit können sowohl boolesche Anfragen als auch Anfragen über das Vektorraummodell verarbeitet werden. Anfragen auf Bildern sind über zugeordnete Textdateien, die eine Beschreibung des Bildes enthalten, oder über den Dateinamen möglich.

Bei Anfragen auf Basis des Vektorraummodells wird die Relevanz eines Dokumentes über die Bestimmung des Winkels zwischen Anfrage- und Dokumentvektor berechnet. Zur Begrenzung des Hauptspeicherbedarfs können auch approximierte Ranking-Werte genutzt werden, die nur eine geringfügige Verschlechterung des Ergebnisses bewirken, aber viel effektiver zu berechnen sind. Die Wörter in der Anfrage und im Index werden standardmäßig auf ihren Wortstamm reduziert (Stemming) sowie Groß- und Kleinschreibung (Case Folding) vereinheitlicht.

Glimpse

Der Indexierer *Glimpse* [MaWu94] benötigt nur einen sehr kleinen Index – die Größe liegt in den meisten Fällen zwischen 2 bis 4% der Textdaten – und bietet boolesches Retrieval, approximative Anfragen wie z.B. das Zulassen von Rechtschreibfehlern im gesuchten Wort und sogar eine Suche mit regulären Ausdrücken.

Die drastische Reduzierung des Speicherbedarfs ist durch einen groben Index möglich, d.h., in der invertierten Liste wird nicht auf Dokumente, sondern auf Blöcke verwiesen. Ein Block umfasst mehrere Dokumente. Die Anzahl der Blöcke ist auf maximal 256 beschränkt, um den Platzbedarf einer Block-Adresse (`BlockID`) gering zu halten. Eine BlockID *b* wird einem Term *t* zugeordnet, falls *t* in einem der Dokumente von *b* vorkommt. In Abbildung 9.6 ist dieser Aufbau zu sehen.

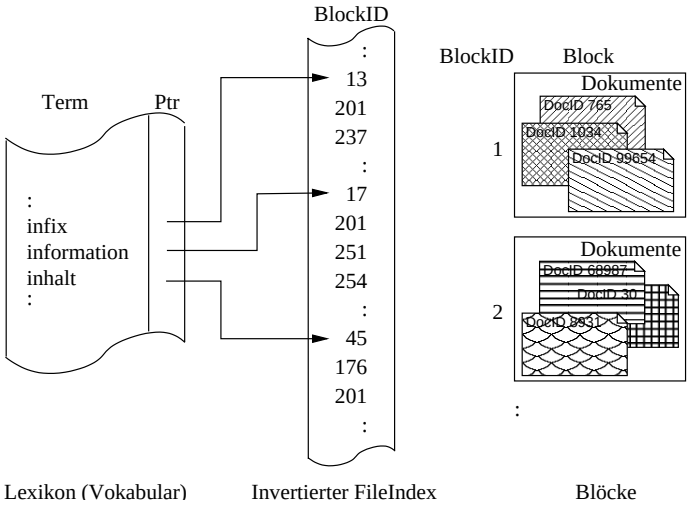


Abb. 9-6: *Aufbau des Glimpse-Indexes*

Der Anfrageprozess besteht aus zwei Phasen (*Two-Level Query Approach*) und stellt einen hybriden Ansatz zwischen einer vollen invertierten Liste und sequenzieller Suche ohne Index dar. In der ersten Phase werden über den Index alle Blöcke ermittelt, die zur Anfrage passende Dokumente enthalten können. Im zweiten Schritt muss dann jeder dieser Blöcke separat durchsucht werden, um die Ergebnisdokumente zu bestimmen. Um eine approximative Suche zu ermöglichen, wird sowohl das Lexikon als auch jeder Block sequenziell mit einem speziellen String-Matching-Algorithmus (*agrep*) durchsucht. Eine Organisation des Lexikons mit Hashing-Verfahren oder B-Bäumen ist in diesem Fall ausgeschlossen, da hier eine exakte Übereinstimmung des Suchterms mit dem Schlüsselwort in der Datenstruktur erforderlich ist (Ausnahme: Präfix- und Bereichssuche im B-Baum).

Boolesche Anfragen werden wie bei regulären invertierten Listen abgearbeitet – je nach Verknüpfung wird entweder der Durchschnitt oder die Vereinigung der

Teillisten gebildet. Bei der AND-Verknüpfung kann es hier allerdings zu so genannten *False Matches* kommen – wenn zwei Terme zwar innerhalb eines Blocks, aber nicht innerhalb eines Dokuments vorkommen, dann qualifiziert sich der Block im ersten Auswertungsschritt, so dass alle Dokumente im Block erfolglos gelesen werden.

Der zweistufige Anfrageprozess führt natürlich dazu, dass Anfragen mit *Glimpse* langsamer sind als bei invertierten Listen mit feineren Granularitäten. Die Beschränkung der Blockanzahl beschränkt außerdem die Größe des zu indexierenden Datenbestandes, der mit zufriedenstellenden Antwortzeiten durchsucht werden kann. Auf der anderen Seite bringt die geringe Indexgröße aber auch Vorteile – zum einen kann der Index schnell konstruiert werden, und zum anderen ist die Modifikation einfach.

9.5 Suchmaschinen in Datenbanken und auf semistrukturierten Dokumenten

Moderne objektrelationale Datenbanken mit einer Texterweiterung können nun die Vorteile von Anfragefunktionalitäten relationaler Datenbanksysteme mit den Fähigkeiten von IR-Systemen verbinden. Im Gegensatz zu den bisher vorgestellten Techniken sind so auch Verknüpfungen verschiedener Dokumente möglich. Semistrukturierte Dokumente können aber weder mit diesen Datenbanken noch mit IR-Systemen geeignet angefragt werden. In diesem Abschnitt wird mit IRQL eine Sprache vorgestellt, die Konzepte beider Bereiche kombiniert und Anfragen auf semistrukturierte Daten ermöglicht. Weitere Schwerpunkte dieses Abschnitts bilden XML-Retrievalsprachen und XML-Suchmaschinen, die IR-Fähigkeiten in XML-Anfragesprachen bzw. XML-Datenbanken integrieren.

9.5.1 Kombination von IR- und Datenbanktechniken

Das Spektrum der im WWW vorhandenen Informationen reicht von semi- oder unstrukturierten Dokumenten bis hin zu strukturierten, in Datenbanken gespeicherten Daten. Die Suche nach bestimmten Informationen gestaltet sich dementsprechend schwierig. Suchmaschinen erlauben zwar inhaltsbasierte Anfragen mit Hilfe von IR-Techniken, können jedoch die Struktur der indexierten Daten im Allgemeinen nicht ausnutzen. Weiterhin werden die in Datenbanken gespeicherten Informationen meistens nicht betrachtet, obwohl gerade hier die Vorteile der jeweiligen Anfragesprache ausgenutzt werden könnten. Auf der anderen Seite sind auch reine Datenbankanfragesprachen wie SQL nicht geeignet, um heterogene, semistrukturierte Dokumente anzufragen. Diese Sprachen bieten zwar Unterstützung für Operationen auf strukturierten Dokumentanteilen, Anfragen an semistrukturierte Teile sind häufig nicht oder nur durch herstellerspezifische Erweiterungen möglich.

Es existieren verschiedene Vorschläge bzw. Implementierungen für Anfragesprachen, die Konzepte aus dem Information-Retrieval- und Datenbankbereich vereinen [FlLM98]. Die Information Retrieval Query Language IRQL [HePr00]

integriert beide Konzepte am stärksten. Das Ziel von IRQL ist die Vereinigung der in Tabelle 9-3 genannten Vorteile von Datenbankanfragesprachen und Information Retrieval in einer Sprache.

Datenbankanfragesprachen	Information Retrieval
Attributierung	inhaltsbasierte Anfragen
typspezifische Informationen	vage Anfragen
Restrukturierung	Ranking
Verknüpfung von Daten	Relevance Feedback

Tab. 9-3: *Vorteile von Datenbankanfragesprachen und Information Retrieval*

IRQL ist eine Anfragesprache im Stil von SQL, die jedoch Möglichkeiten des Information Retrieval durch zusätzliche Klauseln bietet. Die Unterstützung von Anfragen an strukturierte und semistrukturierte Daten erfolgt durch eine Erweiterung des objektrelationalen Datenmodells. Strukturierte Daten werden als Elemente des strukturierten Datentyps `struct` dargestellt und unterliegen damit der statischen Typüberprüfung, semistrukturierte Daten werden als Elemente eines eigenen Datentyps `doc` modelliert. Die Typüberprüfung für Instanzen dieses Datentyps wird so modifiziert, dass Anfragen auch bei unvollständiger Kenntnis des Schemas möglich sind und das erwartete Ergebnis liefern:

- Bei Operationen auf inkompatiblen Daten überführen implizite Type-Casts diese Daten gegebenenfalls in kompatible Typen.
- Anfragen an heterogene Daten können nicht existierende Attribute enthalten, ein geeignetes Ergebnis wird dann in Abhängigkeit von der konkreten Operation geliefert. Nicht existierende Attribute in Projektionslisten werden beispielweise tupelweise ignoriert und bilden daher erneut heterogene Ergebnisse. Bei Selektionsbedingungen, die derartige Attribute referenzieren, ist das Ergebnis `false`.
- Unvollständig bekannte Schemata und deren Instanzen können durch Verwendung von Pfadausdrücken und -variablen angefragt werden.

Das IRQL zugrunde liegende Datenmodell beschreibt die Extension aller bekannten Dokumente. Zur Einschränkung von Anfragen können weitere Extensionen durch (1) die Erreichbarkeit von Dokumenten aufgrund von Pfadinformationen, (2) die Sprache von Dokumenten, (3) den gegebenenfalls benannten Typ von Dokumenten sowie (4) die Domäne von Dokumenten gebildet werden.

Die erforderlichen Operationen für Anfragen an semistrukturierte Daten werden durch Klauseln für inhaltsbasierte Anfragen, Soundex- und Proximity-Suche realisiert. Weitere IR-Eigenschaften werden durch Möglichkeiten der Fragetermgewichtung und des Rankings der Anfrageergebnisse umgesetzt. Diese Anfragemöglichkeiten sind zwar besonders für Anfragen an semistrukturierte Daten geeignet, jedoch nicht darauf beschränkt.

IRQL-Anfragen sind kompatibel zu Datenbankanfragen, falls keine semistrukturierten Daten in der Menge der angefragten Extensionen vorkommen. Auf der anderen Seite wird eine Kompatibilität zu IR-Ausdrücken dadurch erreicht, dass inhaltbasierte Anfragen wie »Datenbanken and IR« akzeptiert und in SQL-Anfragen umgeformt werden.

9.5.2 Retrieval auf XML-Dokumenten

XIRQL

Bei den aktuellen Sprachvorschlägen für XML-Anfragesprachen fehlen die meisten IR-Fähigkeiten, wie Gewichtung und Ranking, relevanzorientierte Suche und Datentypen mit vagen Prädikaten. Die XML IR Query Language XIRQL [FuGr00, FuGr01] integriert diese Eigenschaften durch Nutzung von Ideen aus dem probabilistischen Modell in Kombination mit Konzepten aus dem Datenbankbereich. Die Basis für XIRQL bildet XQL, eine Anfragesprache für XML-Dokumente, die eine natürliche Erweiterung des W3C-Standards XPath darstellt.

XQL unterstützt die strukturorientierte Suche in XML-Dokumenten, bietet aber nur wenige Datentypen wie `Number`, `Date` und `String` und keine vagen Prädikate. Die Sprache lehnt sich sehr eng an die XML-Syntax an. Dies erfordert eine genaue Kenntnis der Struktur des Dokumentbestandes, da es in XML möglich ist, die gleiche Information mit syntaktisch unterschiedlichen Ausdrücken zu formulieren – beispielsweise kann eine bestimmte Information wie etwa ein Autorenname in einem XML-Element oder in einem XML-Attribut kodiert sein.

Die Anfragemöglichkeiten in XQL wurden bereits in Kapitel 3 beschrieben, Details können in [RoLS98] nachgelesen werden. Als Basis für die nachfolgenden Anfragen soll das XML-Dokument in Beispiel 9-2 dienen, dessen Baumstruktur in Abbildung 9-7 zu sehen ist.

```
<book keyword='Datenbanksysteme' year='2000'>
   <author>Andreas Heuer</author>
   <author>Gunter Saake</author>
   <title>Datenbanken: Konzepte und Sprachen</title>
   <chapter>
      <heading>Grundlegende Konzepte</heading>
      Das erste Kapitel ist den grundlegenden ...
   </chapter>
   <chapter>
      <heading>Architekturen von DB-Systemen</heading>
      <section>
         <heading>Systemarchitekuren</heading>
         Ein wesentlicher Aspekt bei Anwendungen ...
      </section>
      <section>
         <heading>Anwendungsarchitekturen</heading>
      </section>
   </chapter>
</book>
```

Beispiel 9-2: *Auszug aus einem XML-Dokument für ein Buch*

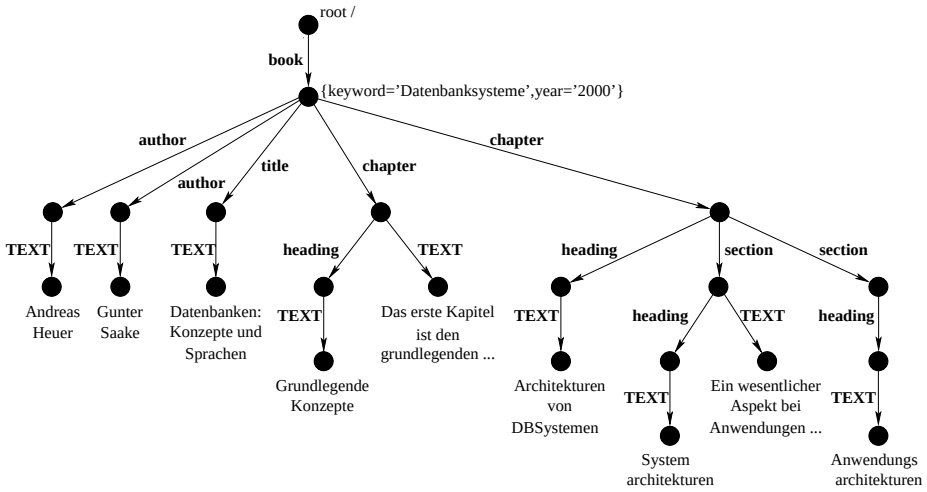


Abb. 9-7: *Baumstruktur des XML-Beispieldokuments*

XIRQL ist eine Erweiterung von XQL um Datentypen und vage Prädikate. Die Sprache unterstützt gewichtete Anfragebedingungen und erlaubt die Abstraktion von der konkreten XML-Syntax, d.h., es kann ignoriert werden, ob die gewünschte Information in einem Attribut oder Element vorkommt. Die XIRQL-Anfrage

```
/book[~keyword approx 'Datenbank' and ~author sounds_like 'Sake']
```

liefert alle Bücher, die ein ähnliches Schlüsselwort wie »Datenbank« haben und deren Autor klingt wie »Sake«. Die Notation `~author` erzwingt, dass nicht zwischen Element und Attribut unterschieden wird. Eine weitere Abstraktion von der konkreten XML-Syntax kann durch die Einführung von Datentypen erreicht werden. Die Suche nach einer Person namens »Heuer« unabhängig von ihrer Rolle (Autor oder Herausgeber) ist dann mit der Anfrage `/book[.//#pname eq 'Heuer']` möglich.

Bei der Suche in strukturierten Volltexten ist es oft nicht klar, welche Strukturebene die Antwort zur Anfrage enthält, z.B. bei der Suche nach einem Buch, in dem »etwas über IR und XML« steht. Falls in einem Buch ein Abschnitt enthalten ist, der beide Worte enthält, so sollte dieser Abschnitt im Anfrageergebnis auftauchen. Die Rückgabe des Abschnitts ist auch dann sinnvoll, wenn beide Wörter in einem Element einer Aufzählungsliste stehen – das Aufzählungselement allein im Ergebnis nützt dem Anwender sicherlich wenig. Kommt aber das Wort IR in Abschnitt 3.2 und das Wort XML in Abschnitt 3.5 vor, dann sollte das gesamte Kapitel 3 zurückgeliefert werden. Die Lösung dieses Problems ist die Zusammenfassung von Strukturelementen zu so genannten Indexknoten. Die Definition dieser Knoten könnte Bestandteil einer erweiterten DTD sein.

Die Indexknoten nehmen die Stellung von Dokumenten in herkömmlichen IR-Systemen ein, d.h., sie bilden die Basis für die Termgewichtung und können als Suchergebnisse zurückgeliefert werden. Für das XML-Dokument in Beispiel 9-1 ist z.B. die Definition der Indexknoten Abschnitt, Kapitel und Buch sinnvoll. Bei zusammengesetzten Anfragen werden nicht nur die spezifischsten Indexknoten zurückgeliefert, sondern auch deren Vorfahren mit reduziertem Retrievalgewicht – bei der Recherche nach einem Buch über »IR und XML« wird der Abschnitt mit beiden Wörtern höher gewichtet als das Kapitel, in dem die gesuchten Wörter in verschiedenen Abschnitten vorkommen.

XXL

Existierende XML-Anfragesprachen unterstützen boolesches Retrieval, d.h., die Anfrageergebnisse sind dabei ungeordnete Mengen von XML-Elementen, die den regulären Suchmustern der Anfrage entsprechen. Dieses Suchparadigma ist für stark schematisierte, »geschlossene« XML-Dokumentkollektionen geeignet. Im Web hingegen ist ein boolesches Suchergebnis wegen der großen Streuung und der daraus resultierenden Unschärfe von Dokumentstrukturen, Vokabular und Dokumentinhalten nicht praktikabel. Hier sollten sich auch XML-Elemente qualifizieren, deren Pfad »ähnlich« zu dem gesuchten Pfad ist und deren Inhalte »ähnlich« sind. Das Ergebnis ist dann eine nach Relevanz absteigend sortierte Rangliste von XML-Elementen. Aus diesen Überlegungen ist die Anfragesprache XXL (flexible XML search language) entstanden [ThWe01, SiTW01], die eine Teilmenge von XML-QL (siehe Kapitel 3) um Ähnlichkeitsvergleiche erweitert.

Im Gegensatz zu XIRQL werden hier nicht nur Ähnlichkeiten zwischen Element- bzw. Attributwerten, sondern auch zwischen den Pfaden von XML-Elementen berücksichtigt. Die Relevanz wird zunächst lokal auf Elementbasis für elementare Ähnlichkeitsbedingungen ermittelt und anschließend zu einem globalen Relevanzmaß für ganze Elementpfade bzw. Teilgraphen des XML-Dokumentgraphen zusammengesetzt. XXL erlaubt die Auswertung semantischer Ähnlichkeitsbedingungen durch den Einsatz von Ontologiegerüsten, die Begriffsbäume darstellen, in denen das für die Anwendungsdomäne spezifische Vokabular manuell bzw. intellektuell modelliert ist. Die Knoten des Baumes werden durch die Begriffe gebildet und die von einem Knoten ausgehenden Kanten zu den Kindern sind geordnet. Kanten drücken Ober-/Unterbegriffsbeziehungen aus, und die Ordnung gibt die semantische Ähnlichkeit zwischen Geschwistern an. Beispielsweise könnten für den Begriff »Datenbank« die Unterbegriffe »Relational«, »Objektrelational« und »Objektorientiert« definiert sein. Durch die Reihenfolge wird ausgedrückt, dass »Relational« enger mit »Objektrelational« verwandt ist als mit »Objektorientiert«. Bei der Suche nach XML-Elementen mit dem Pfad `//Relational` würden sich dann z.B. auch Elemente mit den Pfaden `//Objektrelational` oder `//Datenbank` (mit absteigender Relevanz) qualifizieren.

XML-Suchmaschinen

Existierende XML-Suchmaschinen sind eher mit Volltext-Datenbanksystemen vergleichbar als mit Web-Suchmaschinen, da die XML-Dokumente im Gegensatz zu den Suchmaschinen abgespeichert werden und auch keine Crawler-Komponente mit einer automatischen Erweiterung des Suchraums durch die Weiterverfolgung von Links besitzen. Zwar gibt es vom W3C eine Empfehlung für Links in XML mit der »XML Linking Language (XLink) Version 1.0« [DeMO01], in der Praxis werden solche Links bisher aber kaum verwendet. Die Adressen der XML-Dokumente für die Suchdomäne bzw. die XML-Dokumente selbst müssen einem Indexierer daher manuell übergeben werden. Im Gegensatz zu Volltext-Datenbanksystemen und Web-Suchmaschinen berücksichtigen XML-Suchmaschinen die Struktur der Dokumente und ermöglichen somit »strukturiertes« Information Retrieval [EgLo00], das folgende Erweiterungen bieten kann:

- *Kategorisierung der Dokumente durch das Schema*: Das Schema eines XML-Dokuments identifiziert dessen Struktur und Einsatzzweck. Ergebnisse können so auf Dokumente eingeschränkt werden, die einem bestimmten Schema entsprechen. Beispielweise kann ein Nutzer, der ein Auto kaufen möchte, seine Anfrage auf Schemata eingeschränken, die zum Beschreiben von Verkaufsangeboten für Autos dienen.
- *Unterscheidung von mehrdeutigen Wörtern durch den Kontext, in dem sie erscheinen*: Wörter in natürlicher Sprache haben häufig unterschiedliche Bedeutungen. Der XML-Kontext (einschließende Tags) kann hier genutzt werden, um die gesuchte Bedeutung des Wortes zu klären. Die Suche nach »Albert Einstein« mit einer Web-Suchmaschine liefert beispielweise nach ihm benannte Einrichtungen, Vorlesungen, die seine Theorien beinhalten, und viele andere Seiten, die sich mit ihm in irgendeiner Art und Weise beschäftigen. Interessiert man sich nur für Bücher oder Artikel von Albert Einstein, dann kann das Ergebnis auf XML-Dokumente eingeschränkt werden, in denen Einstein als Autor enthalten ist.
- *Verwendung von Datentypen in Anfragen*, wie z.B. numerische Attribute, Datumsangaben oder andere Werte, deren Semantik schwierig über eine Stichwortsuche formulierbar ist. Da solche Werte in XML von anderen Inhalten getrennt werden, können sie identifiziert und entsprechend ihrer Semantik verwendet werden – die Datentypen können entweder explizit im XML-Schema (deklarierte Datentypen) angegeben sein oder über Heuristiken abgeleitet werden. Bei der Suche nach einem Auto kann das Ergebnis beispielsweise auf solche Angebote eingeschränkt werden, in denen der Angebotspreis unter einer angegebenen Grenze liegt.
- *Nutzung von struktureller Nähe statt physikalischer Nähe zur Bewertung von Anfrageergebnissen*: Information-Retrieval-Systeme nutzen häufig die Nähe der Suchbegriffe in einem Dokument für das Ranking. Die XML-Struktur bietet eine Verbesserung dieses Verfahrens – die Entfernung zwischen dem

letzten Wort in einem Element und dem ersten Wort im nachfolgenden Element ist größer als die Entfernung zwischen benachbarten Wörtern in einem Element, auch wenn die physikalische Nähe ähnlich ist. Sucht ein Nutzer beispielsweise nach einem bestimmten Video-Treiber für Linux über eine einfache Stichwortsuche, dann können ohne Beachtung der Struktur im Ergebnis Dokumente vorhanden sein, die den gesuchten Treiber für Windows und einen anderen Video-Treiber für Linux enthalten. Sind die Treiber dagegen in einem strukturierten XML-Dokument enthalten, dann würde der Windows-Treiber in einem `<driver>`-Element stehen und der Linux-Treiber in einem anderen. Im Suchergebnis würden dann alle Dokumente höher bewertet, die alle Suchbegriffe im gleichen `<driver>`-Element enthalten.

- *Rückgabe relevanter Dokumentbestandteile statt ganzer Dokumente*: Dokumente können oft sehr lang sein, und in vielen Fällen sind nur kleine Anteile aus dem Dokument bezüglich der Nutzeranfrage relevant. Durch die explizit in XML-Dokumenten enthaltene Struktur können diese relevanten Anteile aus dem Dokument extrahiert werden. Als Beispiel soll hier wieder die Suche nach einem Video-Treiber dienen. Ein Dokument kann viele Treiber beinhalten, aber durch die Extraktion der passenden `<driver>`-Elemente kann die Nutzeranfrage direkt beantwortet werden.

XML-Suchmaschinen stellen eine Erweiterung von reinen XML-Datenbanken dar, da sie eine kombinierte Suche über strukturellen und textuellen Informationen gestatten und somit einen Volltextindex benötigen. Als Vertreter von aktuellen XML-Suchmaschinen wird im Folgenden der TEXTML-Server von IXIASOFT genauer vorgestellt.

TEXTML-Server

Der TEXTML-Server [IXIA01] stellt ebenfalls eine XML-Datenbank mit umfangreichen Information-Retrieval-Möglichkeiten dar. XML-Dokumente werden beim Einfügen nicht auf ihre Gültigkeit bezüglich einer DTD oder eines XML-Schemas geprüft – Voraussetzung für ein erfolgreiches Einfügen ist lediglich die Wohlgeformtheit des Dokuments. Beim Einfügen eines XML-Dokumentes wird dessen Inhalt geparst. Anschließend erfolgt eine Extraktion und Indexierung der Werte von Elementen und Attributen entsprechend der definierten Indexstrukturen. Die gebildeten Indexe ermöglichen eine Suche über die extrahierten Werte und dienen als Grundlage für die Formulierung von Anfragen. Die Struktur der Indexe wird vom Administrator definiert, der neben der Angabe der zu indexierenden Elemente und Attribute verschiedene Möglichkeiten zur Einflussnahme auf die Beschaffenheit der Indexe hat:

- Definition spezieller Werte – einzelne Wörter oder Phrasen – für Elemente und Attribute, die in einen Index aufgenommen werden sollen (*String-Index*). Alle anderen vorkommenden Werte werden ignoriert.

- Angabe der Verschachtelungstiefe, bis zu der XML-Elemente berücksichtigt werden sollen.
- Spezifikation von Bedingungen für die Aufnahme von Werten in den Index, z.B. die Angabe von Ober- und Untergrenzen bei einem numerischen Index.

Dadurch lässt sich die Struktur eines Indexes auf einen bestimmten Anwendungsfall »zuschneiden«. Verändern sich die Anforderungen an die Anwendung, dann kann die Struktur und Art der Indexe jederzeit verändert werden. Neben einem Volltextindex (*Wort-Index*) werden auch Indexe für Datums- und Zeitangaben sowie für numerische Werte unterstützt. Der Volltextindex stellt eine feine invertierte Liste dar, d.h. für jeden Term wird nicht nur das Dokument, sondern auch die Position des Terms innerhalb des Dokuments gespeichert. Der Aufbau der Indexe wird in Systemdokumenten festgelegt, die selbst wieder XML-Dokumente sind und in der Datenbank abgelegt werden.

Genauso wie die Indexe selbst lassen sich auch die Zeitpunkte der Indexierung konfigurieren. Die Indexierung kann zu bestimmten Ereignissen gestartet werden. Folgende Ereignisse sind einstellbar: das Einfügen einer bestimmten Anzahl von Dokumenten, das Einfügen einer bestimmten Menge von Informationen (KB), bestimmte feste Zeitpunkte und das Verstreichen einer bestimmten Zeit seit dem letzten Indexierungprozess.

Eine Volltext-Suche kann sowohl auf dem gesamten Dokument als auch auf individuellen Elementen vorgenommen werden. Der TEXTML-Server bietet boolesche Operatoren, Phrasen- und Proximity-Suche, mit der die Entfernung zwischen Suchbegriffen auf eine Anzahl von dazwischen liegenden Wörtern, Elementen und logischen Einheiten wie Sätze und Abschnitte eingeschränkt werden kann. Weiterhin unterstützt der Server Anfragen auf Datums- und Zeitangaben sowie numerischen Werten. Innerhalb von Anfragen können mehrere Indexe unterschiedlichen Typs angegeben werden. Die Anfragesprache des TEXTML-Server basiert auf XML. Die folgende Anfrage ermittelt alle Dokumente mit dem Stichwort »Datenbanksystem« an beliebigen Stellen im Dokument.

```
<Query Version = "2.0" RESULTSPACE = "R1">
   <key NAME = "Full Text">
      <elem>Datenbanksystem<elem>
   </key>
</Query>
```

Beispiel 9-3: *TEXTML: Stichwort-Anfrage*

Eine weitere Suchanfrage soll die Verwendung boolescher Operatoren und Bereichsanfragen auf Datumsangaben verdeutlichen. Gesucht wird hier nach allen Veröffentlichungen des Autors »Heuer« aus den Jahren 1999 bis 2001.

```
<Query Version = "2.0" RESULTSPACE = "R1">
   <andkey>
      <key NAME = "Author Index">
         <elem>Heuer<elem>
      </key>
      <key Name= "Year Index">
         <intervall>
            <start INCLUSIVE = "True">
               <date><year>1999</year></date>
            </start>
            <end INCLUSIVE = "True">
               <date><year>2001</year></date>
            </end>
         </intervall>
      </key>
   </andkey>
</Query>
```

Beispiel 9-4: *TEXTML: Strukturierte Anfrage*

In dieser Anfrage wird eine Stichwortsuche über dem *Author Index*, der einen Volltextindex über dem `author`-Element darstellt, mit einer Bereichssuche über dem Datumsindex *Year Index* verknüpft.

Als Ergebnis wird eine Liste mit den Namen aller Dokumente präsentiert, die der Anfrage entsprechen. Standardmäßig ist diese Liste nach dem Dokumentnamen sortiert. Eine Sortierung ist aber auch nach allen Elementen und Attributen möglich, für die ein Index existiert.

9.6 Zusammenfassung

Das Kapitel hat gezeigt, auf welch vielfältige Arten Suchmaschinen in aktuellen Umgebungen wie Web und XML verwendet werden. Interessant ist hierbei der Umstand, dass viele heute eingesetzte Techniken schon in frühen Forschungsarbeiten und Implementierungen im Bereich des Information Retrieval entwickelt wurden. Populärstes Beipiel für die Anwendung dieser Techniken ist sicherlich das Web. Bestimmte Eigenschaften im Web, wie die enorme Menge und die Schnelllebigkeit der Daten, waren der Grund, warum die IR-Techniken an die Gegebenheiten angepasst werden mussten. Diese Anpassungs- aber auch Neuentwicklungsbestrebungen sind allerdings noch lange nicht abgeschlossen. Gerade die aktuellen Entwicklungen zur besseren Strukturierung der Daten im Web bspw. durch XML und die Probleme beim Suchen im Web belegen die Notwendigkeit weiterer Forschungen in diesem Bereich. Die erste Generation einsetzbarer XML-Suchmaschinen liegt vor. Im Web können diese aber bis jetzt kaum getestet werden, da umfangreiche XML-Dokumentkollektionen praktisch nicht vorhanden sind.

Prinzipiell wird die Trennung der Dokumente in Inhalt, Struktur und Layout ein wichtiger Schritt zur Verbesserung von Suchmaschinen sein, da auf diese Weise eine Konkretisierung von Anfragen auf Strukturaspekte, ein abgestufteres Ranking und die Rückgabe relevanter Dokumentanteile möglich sind.

Die nächste Generation der Web-Suchmaschinen wird über die Möglichkeit verfügen, Datenbanken, Kataloge und andere Dienste bspw. als Web Services einzubinden. Sie selbst werden sich vielleicht ebenfalls als Web Service anbieten und anderen Diensten die Integration ermöglichen. Web Services und deren Techniken werden im nächsten Kapitel (Kap. 10) ausführlich vorgestellt.

Systeme, die Kopplungsmechanismen von Datenbanksystemen, Information-Retrieval-Systemen in XML-Datenbanksystemen entwickelt haben, werden in Kapitel 13 vorgestellt und verglichen.

Literatur

[AVAH00] AltaVista Company. *AltaVista Search Engine 3.0 - Administrator's Handbook*, 2000. http://solutions.altavista.com/docs/AdministratorHandbook _V30_111600.pdf.

[AVDK00] AltaVista Company. *AltaVista Search Engine 3.0 - Search Developer's Kit*, 2000. http://solutions.altavista.com/docs/SearchDeveloperGuide _V30_111600.pdf.

[BaRi99] Ricardo Baeza-Yates, Berthier Ribeiro-Neto. *Modern Information Retrieval*. ACM Press & Addison Wesley, New York, USA, 1999.

[Berg01] Michael K. Bergman. *The Deep Web: Surfacing Hidden Value*. , BrightPlanet Corp., 2001.

[Berk00] *How Much Information? – Internet Summary*. University of California Berkeley, 2000. http://www.sims.berkeley.edu/research/projects/how-much-info/internet.html.

[Call00] Jamie Callan. *Distributed Information Retrieval*. In W. Bruce Croft, editor, Advances in Information Retrieval – Recent Research from the Center for Intelligent Information Retrieval, pp. 127–150. Kluwer Academic Publishers, Boston, 2000.

[DeMO01] Steve DeRose, Eve Maler, David Orchard. *XML Linking Language (XLink) Version 1.0*. W3C Recommendation, W3C, 2001. http:// www.w3.org/TR/2001/REC-xlink-20010627/.

[EgLo00] Daniel Egnor, Robert Lord. *Structured Information Retrieval using XML*. In Proceedings of the ACM SIGIR 2000 workshop on XML and Information Retrieval, 2000.

[FlLM98] Daniela Florescu, Alon Y. Levy, Alberto O. Mendelzon. *Database Techniques for the World-Wide Web: A Survey. SIGMOD Record*, 27(3): 59–74, 1998.

[FuGr00] Norbert Fuhr, Kai Großjohann. *XIRQL – An Extension of XQL for Information Retrieval*. In Proceedings ACM SIGIR 2000 Workshop on XML and Information Retrieval. ACM, 2000.

[FuGr01] Norbert Fuhr, Kai Großjohann. *XIRQL: A Query Language for Information Retrieval in XML Documents*. In Proceedings of the 24th Annual International Conference on Research and Develop*ment in Information Retrieval*, pp. 172–180. ACM, 2001.

[HSWL01] Darren R. Hardy, Michael F. Schwartz, Duane Wessels und Kang-Jin Lee. *Harvest User Manual - Harvest Version 1.7*. , 2001.

[HePr00] Andreas Heuer, Denny Priebe. *Integrating a Query Language for Structured and Semi-Structured Data and IR Techniques*. In Proceedings of the 11th Intl. Workshop on Database and Expert System Applications. IEEE Computer Society, 2000.

[IMG01] INT Media Group Inc. *Search Engine Watch – Ratings, Reviews and Tests*, 2001. http://www.searchenginewatch.com/reports/index.html.

[ISO999] *ANSI/ISO/IEC International Standard (IS) Database Language SQL – Part 1: SQL/Framework, ISO/IEC 9075-1: 1999 (E)*, 1999.

[IXIA01] IXIASOFT. *TEXTML-Server*. White paper, IXIASOFT, 2001. http://www.ixiasoft.com/support/doc/ixia_WP.pdf.

[Klei98] J. M. Kleinberg. *Authoritative sources in a hyperlinked environment*. In Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithmns, San Francisco, California, 1998.

[Kl+01] Meike Klettke et al. *GETESS - Ontologien, objektrelationale Datenbanken und Textanalyse als Bausteine einer semantischen Suchmaschine*. In Datenbank Spektrum – Zeitschrift für Datenbanktechnologie, 1: 14–24, 2001.

[LaGi98] Steve Lawrence, C. Lee Giles. *Searching the World Wide Web*. *Science*, 280(5360): 98–100, 1998.

[LaGi99] Steve Lawrence, C. Lee Giles. *Accessibility of information on the web*. In Nature, (400), July 1999.

[MaWu94] U. Manber, S. Wu. *GLIMPSE: A Tool to Search Through Entire File Systems*. In Proceedings of the USENIX Winter 1994 Technical Conference, pp. 23–32, San Fransisco, USA, 994.

[OCLC01] Inc. Office of Research OCLC Online Computer Library Center. Web Characterization – Size and Growth, 2001. http://wcp.oclc.org/.

[Page98] L. Page. *The pagerank citation ranking: bringing order to the web*. In Proceedings of the Annual Meeting of the American Society for Information Science, 1998.

[RaGa01] Sriram Raghavan, Hector Garcia-Molina. *Crawling the Hidden Web*. In Peter M. G. Apers et al., editors, Proceedings of 27th International Conference on Very Large Data Bases, pp. 129–138. Morgan Kaufmann Publishers, San Francisco, 2001.

[Rape01] Joaquin Rapela. *Automatically Combining Ranking Heuristics for HTML Documents*. In Proceedings of the WIDM2001, Atlanta, USA, 2001.

[RoLS98] Jonathan Robie, Joe Lapp, David Schach. *XML Query Language (XQL)*. In M. Marchiori, editor, QL'98 – The Query Languages Workshop, 1998. http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[SiTW01] Sergej Sizova, Anja Theobald, Gerhard Weikum. *Ähnlichkeitssuche auf XML-Daten*. In Andreas Heuer, Frank Leymann und Denny Priebe, editors, Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), Informatik Aktuell, Seiten 364–383. Springer, 2001.

[ThWe01] Anja Theobald, Gerhard Weikum. *Adding Relevance to XML*. In The World Wide Web and Databases, Third International Workshop WebDB 2000, 1997: 105–124, 2001.

[VR79] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow, 1979.

[Webe01] Gunnar Weber. *Integration von Datenbanken in Suchmaschinen bei unterschiedlichen Kooperationsgraden*. In K. Bauknecht et al., editors, Informatik 2001: Wirtschaft und Wissenschaft in der Network Economy - Visionen und Wirklichkeit, volume I, Seiten 345–352. Österreichische Computer Gesellschaft, 2001.

[WiMB99] Ian H. Witten, Alistair Moffat, Timothy Bell. *Managing Gigabytes Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, San Fransisco, 2. Auflage, 1999.

10 Web Services

Markus Keidl, Alfons Kemper, Stefan Seltzsam, Konrad Stocker

Kurzfassung

In diesem Kapitel werden die wichtigsten Standards und Technologien im Bereich Web Services vorgestellt. Web Services verwandeln das Internet immer mehr zu einer Plattform, auf der Dienste angeboten werden. Sie ermöglichen vollautomatische Dienstnutzung und Dienstkomposition. Anhand eines einfachen Beispieldienstes, einem Temperaturdienst für die Stadt Passau, werden die wichtigsten Standards im Bereich Web Services erklärt, die allesamt auf XML basieren: SOAP für den Datenaustausch, UDDI und WS-Inspection für die Dienstverwaltung und WSDL für die Dienstbeschreibung. Neben diesen grundlegenden Standards wird aufbauend auf dem Beispieldienst gezeigt, wie Web Services die Komposition von neuen, zusammengesetzten Diensten unter Verwendung von bestehenden Diensten erleichtern und wie damit die Entwicklung komplexer Dienste unterstützt wird. Außerdem wird das ServiceGlobe-System als eine Beispielplattform für Web Services beschrieben.

10.1 Einleitung

In den letzten Jahren hat sich das Internet immer mehr zu einer Plattform entwickelt, auf der Dienste angeboten werden. Bisher verwenden Dienste im Internet meist eine Kombination aus HTML-Seiten und HTML-Formularen als Schnittstelle, da sie für die Anzeige in einem Browser und die Interaktion mit einem menschlichen Benutzer konzipiert wurden. Aus Gründen der Effizienzsteigerung wollen aber viele Firmen mittlerweile Dienste automatisiert nutzen, also ohne menschliche Interaktion, und eigene Dienste schnell und unkompliziert über das Internet zur Verfügung stellen. Für diesen Zweck sind Formulare ungeeignet, da für jeden Dienst eigene, spezifische Formulare entwickelt werden müssen. Dies erschwert zum einen das Bereitstellen von Diensten und zum anderen deren automatisierte Nutzung und die Ermittlung von Ergebnissen oder Fehlermeldungen aus den angezeigten HTML-Seiten. Zur Ermittlung der interessanten Informationen einer HTML-Seite müssen aufwändige »Screen scraping«-Techniken eingesetzt werden. Diese sind gegenüber Änderungen des Designs der HTML-Seiten nicht robust und müssen so immer wieder angepasst werden.

Um vollautomatische Dienstnutzung und Dienstkomposition (Interoperabilität) zu ermöglichen, wird derzeit immer öfter eine neue Technologie eingesetzt:

Web Services (im Folgenden auch Web-Dienste oder Dienste genannt). Viele große Softwarefirmen haben mittlerweile entsprechende Produkte im Angebot oder entwickeln gerade ihre eigene Web-Service-Lösung. Als die bekanntesten Vertreter sind hier sicherlich BEA WebLogic, HP Web Services Platform, IBM WebSphere, Microsoft .NET, mySAP.com von SAP und SUN One zu nennen.

Bisher gibt es keine einheitliche Definition des Begriffs Web Service. Im üblichen Sprachgebrauch bezeichnet ein Web Service einen Dienst, der Benutzern über das Web zur Verfügung gestellt wird und dabei bspw. auf XML [KeEi01] und HTTP zurückgreift. Web Services unterscheiden sich dabei von klassischen Diensten im Web dadurch, dass sie nicht auf die Benutzung durch Menschen, sondern auf eine automatisierte Benutzung ausgerichtet sind. Ein weiteres Ziel von Web Services ist die Interoperabilität, das heißt, Web Services sollen unabhängig vom Betriebssystem, unabhängig von der Programmiersprache, in der die Services entwickelt worden sind, und unabhängig von der Web Service Engine – so bezeichnet man eine Anwendung, die Web Services in einem Netzwerk verfügbar macht – in einer standardisierten Weise genutzt werden und auch miteinander interagieren können.

Ein bekanntes Beispiel für Web Services sind Marktplätze, z. B. Marktplätze der Automobilindustrie, die dazu dienen, den Einkauf teilnehmender Automobilfirmen zu optimieren und zu automatisieren, d. h. möglichst schnell das günstigste Produktangebot aus den Angeboten aller Zulieferer zu finden. Marktplätze automatisieren auch den Bestellvorgang und die Abrechnung, was zu weiteren Einsparungen und schnellerer Abarbeitung führt. Um dies zu leisten, integrieren Marktplätze die Daten und ERP-Systeme (Enterprise-Resource-Planning-Systeme) der beteiligten Anbieter in ein zentrales System. [KeWi01, WiWK02] beschreiben eine alternative Architektur für Marktplätze, die keine vollständige Datenintegration am Marktplatz erfordert. Sie erlaubt Anbietern, Teile der Daten in ihren lokalen Systemen zu belassen, und ermöglicht es dadurch beispielsweise, Preise sehr dynamisch zu kalkulieren.

Andere bedeutende Anwendungsbereiche von Web Services sind unter anderem Anwendungsintegration, elektronischer Datenaustausch, Electronic-Business-Anwendungen und Business-to-Business-Integration. Für derartige Anwendungen sind Qualitätskriterien und -garantien, z. B. Antwortzeitgarantien [KSWD02], ein wichtiger Aspekt, den wir hier allerdings nicht betrachten werden, da er über den Rahmen dieses Kapitels hinausgeht.

Um die Interoperabilität von Diensten zu gewährleisten, sind Standards nötig. Mittlerweile existieren mehrere verschiedene, auf XML basierende Standards: SOAP (Simple Object Access Protocol von IBM, Microsoft, …), ebXML (Electronic Business using eXtensible Markup Language von OASIS und UN/CEFACT), UDDI (Universal Description, Discovery and Integration von HP, IBM, Intel, Microsoft, SAP, Software AG, Sun, …), WSDL (Web Services Description Language von Ariba, IBM und Microsoft), WSFL (Web Services Flow Language von IBM), XLANG (Microsoft), WS-Inspection (Web Service Inspection Language von IBM und Microsoft) und einige andere. Die Bedeutung der Interoperabilität zeigt sich auch in Firmenkooperationen, die es sich zur Aufgabe machen, Referenzarchitek-

turen basierend auf den bestehenden Standards zu entwickeln. Ein Beispiel hierfür ist WS-I[1] (Web Services Interoperability Organization), zu deren Mitgliedern unter anderem Accenture, DaimlerChrysler und SAP zählen.

Wir werden in diesem Kapitel Web-Dienste exemplarisch auf der Basis von SOAP, UDDI, WS-Inspection und WSDL erläutern, da diese Standards schon sehr weit verbreitet sind und auch die meisten der oben genannten Web-Service-Lösungen darauf basieren oder diese Protokolle zumindest unterstützen. Grundlage für die Interoperabilität ist ein einheitliches Kommunikationsprotokoll (in unserem Fall SOAP), das es ermöglicht, auf eine standardisierte Weise mit Diensten zu kommunizieren. Es ist natürlich nicht ausreichend, Dienste nur zur Verfügung zu stellen, man muss es potenziellen Nutzern auch ermöglichen, diese Dienste zu finden. Bisher wurden Informationen im Internet durch Suchmaschinen oder Web-Kataloge indexiert und damit auffindbar gemacht. In der Welt der Web Services übernehmen diese Aufgaben beispielsweise WS-Inspection und UDDI. Hat man einen Dienst gefunden, benötigt man noch die Information, welche Eingabedaten der Dienst erwartet und wie er sein Ergebnis zur Verfügung stellt. Diese Dienstbeschreibung kann beispielsweise mit Hilfe von WSDL erfolgen.

10.2 Beispiel-Szenario

Um die Protokolle und Abläufe beim Anbieten und Nutzen von Web Services nicht nur abstrakt darstellen zu können, verwenden wir einen Temperaturdienst als Beispiel-Szenario und zeigen anhand dieses Dienstes, was ein Anbieter bzw. ein Benutzer eines Web Service tun muss und wie die verschiedenen Protokolle verwendet werden. Informationen zu den in diesem Kapitel vorgestellten Diensten (inklusive aller notwendigen XML-Dokumente) bieten wir über unseren Server `wetter.fmi.uni-passau.de` an, auf dem diese Dienste auch betrieben werden.

Der Temperaturdienst liefert zu einem vom Benutzer vorgegebenen Zeitpunkt die Temperatur zurück, die an einem bestimmten Temperatursensor in der Stadt Passau gemessen wurde. Hat der Dienst für den geforderten Zeitpunkt keine Messdaten zur Verfügung, liefert er die Temperatur zum nächstgelegenen Zeitpunkt zurück, zu dem ein Messwert existiert. Weitere Dienste in unserem Beispiel-Szenario sind ein *Einheitenumrechnerdienst* und ein *TemperaturInFahrenheit-Dienst*. Der Einheitenumrechnerdienst rechnet Temperaturangaben in verschiedene Einheiten um, der TemperaturInFahrenheitDienst liefert wie der Temperaturdienst die Temperatur in der Stadt Passau zu einem gegebenen Zeitpunkt, allerdings in Grad Fahrenheit.

1. Siehe `www.ws-i.org`

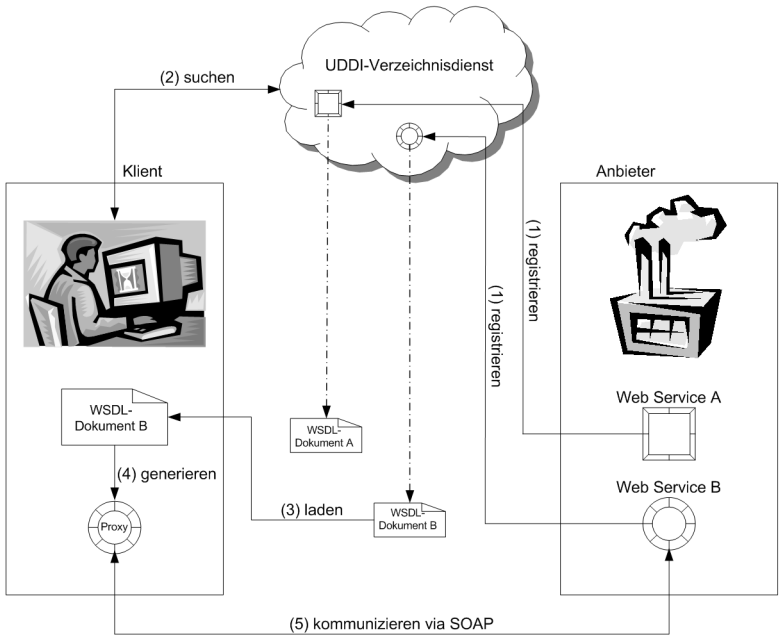


Abb. 10-1: *Überblick über den Einsatz von Web Services*

Abbildung 10-1 gibt einen Überblick über alle nötigen Aktionen, um einen Web Service zur Verfügung zu stellen und diesen zu nutzen. Details zu diesen Schritten findet man in den jeweiligen Abschnitten dieses Kapitels. Ein Dienstanbieter muss seine Dienste (im Bild Web Service A und Web Service B) bei einem UDDI-Verzeichnisdienst registrieren, um die Informationen verfügbar zu machen, welche Dienste er anbietet und wie man diese ansprechen kann. Wie die Kommunikation mit den Diensten aussehen muss, ist dabei in WSDL-Dokumenten beschrieben, die nicht im UDDI-Verzeichnis selbst, sondern »irgendwo« im Internet gespeichert sind. Will nun ein Client einen Dienst nutzen, sucht er sich einen für seine Zwecke geeigneten Dienst aus dem UDDI-Verzeichnis aus. Nach der Auswahl eines Dienstes (in der Abbildung Web Service B), wird das zugehörige WSDL-Dokument geladen und dazu verwendet, einen Proxy für den Web Service zu generieren. Diese beiden Schritte können bereits automatisiert werden. Der generierte Proxy wird verwendet, um mit dem tatsächlichen Web Service via SOAP-Nachrichten zu kommunizieren. Die gerade aufgezählten Schritte werden nun – zum einen aus der Sicht des Dienstanbieters, zum anderen aus Clientsicht – detaillierter beschrieben.

Abbildung 10-2 zeigt eine Übersicht über die Aktionen, die ein Anbieter ausführen muss, damit er einen Dienst anbieten kann. Zuerst benutzt der Anbieter einen UDDI-Verzeichnisdienst, um zu überprüfen, ob es bereits ein so genanntes *tModel* (siehe Abschnitt 10.4.1) gibt, das die Art von Dienst beschreibt, die er anbieten will. Sollte dies der Fall sein, verwendet er dieses tModel inklusive des zugeordneten WSDL-Dokumentes als Grundlage für seinen Dienst. Sollte kein geeignetes tModel existieren, muss der Dienstanbieter ein neues tModel und ein

dazugehörendes WSDL-Dokument erzeugen und bei dem UDDI-Verzeichnisdienst registrieren. Auf der Basis des WSDL-Dokumentes kann sich der Dienstanbieter durch ein geeignetes Werkzeug ein Gerüst des Web Service generieren lassen, also beispielsweise eine Java-Klasse. Diese Klasse genügt dann bereits der Schnittstelle, die im WSDL-Dokument festgelegt ist. Nun muss der Dienstbetreiber das Gerüst noch vervollständigen, also die Funktionalität des Dienstes implementieren. Danach kann er den Web Service betreiben. Damit der Dienst auch von anderen gefunden werden kann, muss er anschließend noch bei einem UDDI-Verzeichnisdienst registriert werden.

Der gerade skizzierte Weg beschreibt nur eine Möglichkeit, einen Dienst im Netz verfügbar zu machen. Wenn beispielsweise eine Anwendung bereits existiert und als Dienst verfügbar gemacht werden soll, kann auch ein WSDL-Dokument aus dem vorhandenen Code generiert werden. Für dieses WSDL-Dokument kann man dann ein entsprechendes tModel beim UDDI-Verzeichnisdienst registrieren.

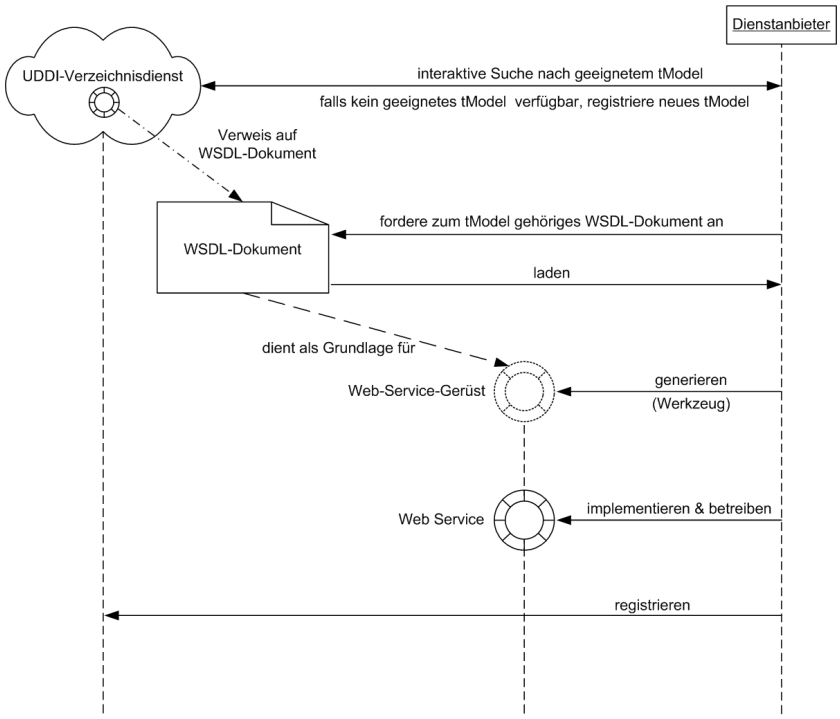


Abb. 10-2: *Aktionen des Dienstanbieters*

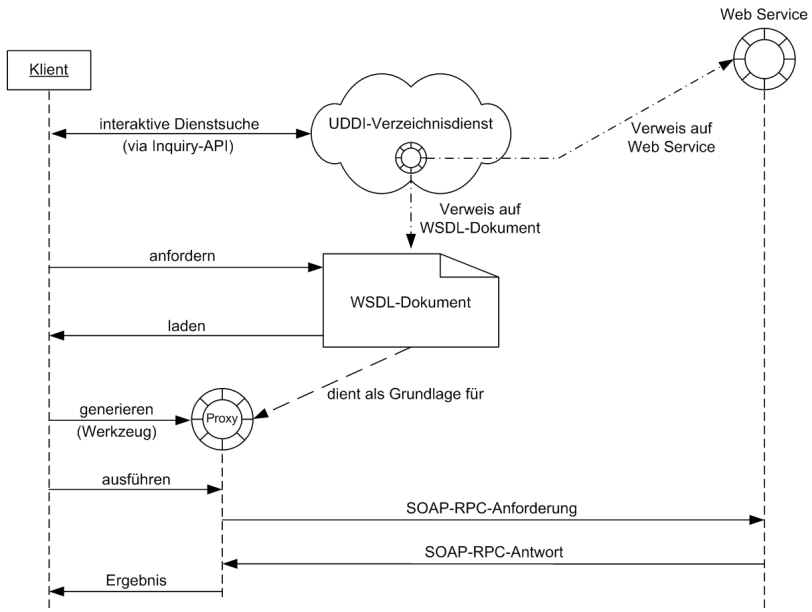


Abb. 10-3: *Aktionen von Dienstbenutzern*

Abbildung 10-3 zeigt, ähnlich einem Sequenzdiagramm, die Aktionen, die Clients ausführen müssen, um einen Dienst nutzen zu können. Wie bereits erwähnt suchen sie dafür nach einem geeigneten Dienst in einem Dienstverzeichnis. Dazu sprechen sie dieses Verzeichnis mit Hilfe des so genannten Inquiry-API an. Wurde ein Dienst gefunden, kann das zugehörige WSDL-Dokument aus dem Internet geladen werden. Wo dieses Dokument zu finden ist, ist im UDDI-Verzeichnis abgelegt. Das WSDL-Dokument spezifiziert, wie Nachrichten an den ausgewählten Dienst aussehen müssen. Im UDDI-Verzeichnis steht, unter welcher URL der Dienst im Internet erreichbar ist. Unter Verwendung eines geeigneten Werkzeugs kann man aus den Informationen des WSDL-Dokumentes und der URL aus dem UDDI-Verzeichnis einen Proxy für die Interaktion mit dem Web-Dienst generieren lassen. Dies kann zum Beispiel eine Java-Klasse mit einer Methode sein, die alle für den Aufruf des Dienstes erforderlichen Parameter übergeben bekommt. Der Proxy kümmert sich dann darum, dass diese Parameter im richtigen Format in eine SOAP-RPC-Nachricht (siehe 10.3.2) verpackt an den Web Service geschickt werden. Außerdem verarbeitet der Proxy das Ergebnis des Aufrufes und wandelt es beispielsweise in ein Java-Objekt um. Auf diese Weise verbirgt der Proxy, dass überhaupt ein Web Service benutzt wird.

Im Folgenden werden die verschiedenen erwähnten Standards anhand des Temperaturdienst-Beispiels detaillierter beschrieben. Der nächste Abschnitt erklärt das Kommunikationsprotokoll SOAP, das von Diensten zum Datenaustausch verwendet wird. Anschließend werden in Abschnitt 10.4 zwei Möglichkeiten zur Dienstverwaltung beschrieben: UDDI als zentrales Dienstverzeichnis und WS-Inspection als Möglichkeit, auf einem Web-Server in definierter Weise auf lo-

kale Dienste zu verweisen. Abschnitt 10.5 erläutert, wie Dienste mit Hilfe von WSDL beschrieben werden können. Der darauf folgende Abschnitt beschreibt die Dienstkomposition und –interaktion, d. h. wie neue, zusammengesetzte Dienste basierend auf existierenden Diensten entwickelt werden können. In Abschnitt 10.7 wird kurz auf verschiedene Plattformen, Produkte und Infrastrukturen für Web Services eingegangen und als Beispiel für eine Dienstplattform der Forschungsprototyp ServiceGlobe vorgestellt. Abschnitt 10.8 beendet dieses Kapitel mit einer Zusammenfassung und einem Ausblick.

10.3 Datenaustausch (SOAP)

SOAP (Simple Object Access Protocol) ist ein Kommunikationsprotokoll für verteilte Anwendungen, das es ermöglicht, strukturierte und typisierte Daten mit Hilfe von XML auszutauschen. Der große Vorteil von SOAP besteht darin, dass verschiedene Protokolle wie z. B. HTTP, SMTP (Simple Mail Transfer Protocol) und FTP (File Transfer Protocol) als Übertragungsprotokoll verwendet werden können. Damit ist es mit SOAP sogar möglich, durch die meisten Firewalls hindurch Nachrichten zu verschicken, was bei anderen Protokollen im Bereich verteilter Anwendungen wie z. B. IIOP (Internet Inter-ORB Protocol) normalerweise nicht möglich ist.[2]

Für den Datenaustausch definiert das SOAP-Protokoll, wie Objekte serialisiert werden, also wie Objekte bzw. vernetzte Objektstrukturen (sequenziell) auf XML abgebildet werden und umgekehrt. SOAP definiert selbst keine Anwendungssemantik und kann dadurch für ein breites Anwendungsspektrum vom einfachen Zustellen einer Nachricht bis zum entfernten Prozeduraufruf (Remote Procedure Call; RPC) verwendet werden. Wir beschränken uns aus Platzgründen auf die Beschreibung der grundlegenden Funktionalität von SOAP – weitergehende Informationen findet man beispielsweise in der SOAP-Spezifikation [BEKL00] oder in dem Buch [ScSt00].

10.3.1 Nachrichtenformat

Der Aufbau einer SOAP-Nachricht sieht folgendermaßen aus:

```
<soap:Envelope  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <!--Der Header ist optional -->
  </soap:Header>

<soap:Body>
    <!-- Serialisierte Objektdaten -->
  </soap:Body>
</soap:Envelope>
```

SOAP-Dokument 10-1: *Aufbau einer SOAP-Nachricht*

2. Allerdings entstehen dadurch auch Sicherheitsrisiken, die durch die bisher verfügbaren Firewall-Lösungen nicht hinreichend abgedeckt werden.

Das Envelope-Element ist das Wurzelelement der Nachricht und kann neben einem Body-Element ein optionales Header-Element enthalten. Durch das encodingStyle-Attribut kann angegeben werden, wie Objekte zu XML serialisiert worden sind, um in dieser Nachricht verschickt zu werden. Dieses Attribut kann auch innerhalb anderer Elemente verwendet werden und gilt dann nur für den entsprechenden Teil der Nachricht. Der im Beispiel angegebene encodingStyle entspricht der Standard-Serialisierung von SOAP (also der Serialisierung, die in der Spezifikation angegeben ist), er muss aber trotzdem angegeben werden. Außerdem wird in dem Envelope-Element noch das Kürzel soap für den SOAP-Namensraum definiert.

Der Header einer SOAP-Nachricht bietet einen generischen Mechanismus, um SOAP dezentral erweitern zu können, ohne diese Erweiterungen vorher mit anderen Kommunikationspartnern abstimmen zu müssen. SOAP definiert einige Attribute, die es erlauben anzugeben, wie der Empfänger mit Header-Erweiterungen umgehen soll und ob diese Erweiterungen verpflichtend verstanden werden müssen. Beispiele für derartige Erweiterungen sind z. B. Authentifizierung, Abrechnung oder Transaktionsmanagement. Ein weiteres Beispiel ist die Web Service Security Language [ABDK02], die SOAP-Nachrichten mit sicherheitsrelevanten Informationen anreichert. Die Möglichkeiten des Header-Elementes gehen allerdings über den Rahmen dieser SOAP-Einführung hinaus. Die interessierten Leser werden auf weiterführende Literatur verwiesen [BEKL00, ScSt00]. Die SOAP-Nachrichten in diesem Abschnitt beinhalten der Einfachheit halber kein Header-Element.

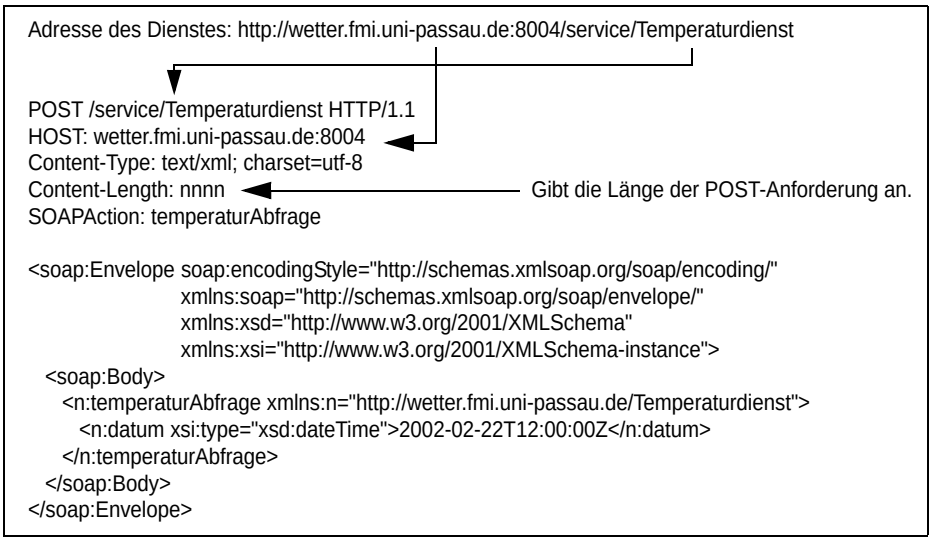
Das Body-Element einer SOAP-Nachricht bietet eine einfache Möglichkeit, Daten zwischen dem Sender und dem Empfänger auszutauschen. Typische Verwendungszwecke des Body-Elementes sind die Aufnahme von serialisierten Daten für RPC-Aufrufe oder von Fehlerbenachrichtigungen. SOAP selbst definiert nur ein Element, das innerhalb des Body-Elementes vorkommen kann: das Fault-Element. Dieses Element wird zur Übermittlung von Fehlerzuständen genutzt und beinhaltet weitere Unterelemente, deren Beschreibung über den Rahmen dieser Einführung hinausgeht.

Die Standard-Serialisierung von Daten basiert auf einem einfachen Typsystem, das eine Generalisierung der gebräuchlichen Typsysteme von Programmiersprachen, Datenbanken und semistrukturierten Datenmodellen darstellt. Ein Typ ist dabei entweder ein einfacher (skalarer) Typ, z. B. String oder Integer, oder ein zusammengesetzter Typ, z. B. Adresse. SOAP legt unter anderem fest, wie Arrays und Referenzen abgebildet werden und wie komplette Graphen von Datenobjekten dieses Typsystems in XML abgebildet werden und umgekehrt. Die Abbildung vom Typsystem einer Programmiersprache in das Typsystem, das der Standard-Serialisierung zugrunde liegt, ist nicht spezifiziert und muss für Typen, die über die Typen der Standard-Serialisierung hinausgehen, festgelegt werden. Eine detaillierte Beschreibung der SOAP-Standard-Serialisierung findet man in der SOAP-Spezifikation [BEKL00].

10.3.2 Kommunikation mit SOAP

Der SOAP-Standard legt nicht fest, welches Protokoll zum Versenden von Nachrichten verwendet werden soll, gibt aber als eine standardisierte Möglichkeit die Einbettung von SOAP in HTTP an. Dabei werden HTTP-POST-Anforderungen[3] für den Transport von SOAP-Nachrichten an einen Server verwendet, und HTTP-Antworten liefern das Ergebnis an den Client zurück. SOAP-Nachrichten könnten allerdings auch mit Hilfe eines Mailprotokolls (SMTP) oder eines Dateiübertragungsprotokolls (FTP) transportiert werden. Diese Möglichkeiten sind bisher allerdings nicht standardisiert.

Die HTTP-Einbettung von SOAP nutzt die vielfältigen Möglichkeiten von HTTP, ohne dabei dessen Semantik zu ändern. Der bestehende Standard wird lediglich dazu benutzt, SOAP-Nachrichten als Nutzlast zu transportieren. Außerdem wird durch das SOAP-Protokoll festgelegt, wie SOAP (zusammen mit der HTTP-Einbettung) für RPC-Aufrufe genutzt werden kann. Dabei werden die URI des Zielobjektes und die Methode, die aufgerufen werden soll, spezifiziert und die Parameter für die Methode übergeben. Nachfolgend zeigen wir zwei Nachrichten eines SOAP-RPC-Aufrufs unseres Temperaturdienstes. Die SOAP-Dokumente sind in eine HTTP-POST-Anforderung bzw. eine HTTP-Antwort eingebettet. SOAP-Dokument 10-2 zeigt die Nachricht, die an den Temperaturdienst geschickt wird, und gibt an, wie die Felder des HTTP-Headers belegt sind.



SOAP-Dokument 10-2: *SOAP-Nachricht, eingebettet in eine HTTP-POST-Anforderung*

3. Eine HTTP-POST-Anforderung ist eine spezielle Nachricht des HTTP-Protokolls, die zur Übertragung von größeren Datenmengen genutzt wird.

Das Feld SOAPAction gibt die Bestimmung der SOAP-Nachricht an. Hier kann eine beliebige URI stehen, in unserem Fall ist es der Name der Methode, die aufgerufen werden soll. Das Envelope-Element spezifiziert die Namensräume und die verwendete Serialisierung für das Dokument. Das Body-Element enthält ein Unterelement, das genauso heißen muss wie die Methode, die aufgerufen werden soll, also temperaturAbfrage. Innerhalb dieses Elementes werden alle Parameter in derselben Reihenfolge und mit demselben Namen aufgezählt, wie in der Methodendeklaration angegeben. Bei unserem Temperaturdienst-Beispiel ist dies nur ein Parameter mit dem Namen datum (vom Typ dateTime). Insgesamt ist diese SOAP-Nachricht also eine Anfrage an den Temperaturdienst, die am 22.02.2002 um 12.00 Uhr gemessene Temperatur zu liefern.[4]

Die Antwort des Dienstes ist in dem SOAP-Dokument 10-3 dargestellt: Wie der HTTP-Header anzeigt (HTTP/1.1 200 OK), wurde die Anforderung erfolgreich bearbeitet. Die Antwort des Dienstes findet man innerhalb des Body-Elementes. Der Name des »Antwortelementes« wird üblicherweise aus dem Namen des »Abfrageelementes« und einem angehängten »Response« gebildet, in unserem Fall also temperaturAbfrageResponse. Das Ergebnis der Abfrage wird üblicherweise innerhalb eines Elementes mit dem Namen Result zurückgegeben. Die beiden Elementnamen sind allerdings zur Auswertung des Ergebnisses nicht relevant. Wie in der Antwort zu sehen ist, wurden am 22.02.2002 um 13.00 Uhr (also dem nächstgelegenen verfügbaren Messtermin zum 22.02.2002, 12.00 Uhr) 4,4 Grad Celsius gemessen.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: nnnn

<soap:Envelope  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soap:Body>
  <n:temperaturAbfrageResponse xmlns:n="http://wetter.fmi.uni-passau.de/Temperaturdienst">
    <Result   xsi:type="ns2:TemperaturdienstAntwort"
              xmlns:ns2="http://wetter.fmi.uni-passau.de/Temperaturdienst.xsd">
      <temperatur xsi:type="xsd:double">4.4</temperatur>
      <einheit xsi:type="xsd:string">Celsius</einheit>
      <messdatum xsi:type="xsd:dateTime">2002-02-22T13:00:00Z</messdatum>
    </Result>
  </n:temperaturAbfrageResponse>
 </soap:Body>
</soap:Envelope>
```

SOAP-Dokument 10-3: *SOAP-Nachricht, eingebettet in eine HTTP-Antwort*

4. Das »Z« am Ende der Datumsangabe in der SOAP-Nachricht zeigt an, dass Coordinated Universal Time (UTC, früher auch Greenwich Mean Time – GMT – genannt) verwendet wird.

10.4 Dienstverwaltung

Um die Nutzung von Diensten zu ermöglichen, müssen Informationen über angebotene Dienste gefunden werden können. Diesen Zweck erfüllen Verzeichnisdienste für Dienstinformationen. Die UDDI-Initiative (Universal Description, Discovery and Integration) [UDDI] hat zum Ziel, ein globales Verzeichnis für solche Dienst-Metadaten zu etablieren. Dieser Initiative haben sich bereits mehr als 300 Firmen angeschlossen. Darunter befinden sich Branchengrößen wie HP, IBM, Microsoft, SAP und Software AG.

Eine Aufgabe des UDDI-Verzeichnisdienstes ist die Speicherung von Dienst-Metadaten in einer einheitlichen Datenstruktur (UDDI-Schema) an zentralen und öffentlich zugänglichen Stellen im Internet (UDDI-Server) durch einheitliche Veröffentlichungs-Mechanismen (UDDI-Publishing-API). Eine weitere Aufgabe ist die Unterstützung der Metadatenabfrage durch eine normierte Anfragesprache (UDDI-Inquiry-API).

Dienstanbieter können Metadaten ihrer Dienste auch – alternativ oder zusätzlich zu der Speicherung in einem globalen UDDI-Verzeichnis – in standardisierter Form auf ihrem Web-Server anbieten, indem sie dort entsprechende Dateien (WS-Inspection-Dokumente) ablegen. Bei Inspektion des Web-Servers, beispielsweise durch eine Suchmaschine, können diese Dateien, die Verweise auf Informationen über die verfügbaren Dienste eines Anbieters enthalten, ausgewertet und einem Benutzer verfügbar gemacht werden (siehe Abschnitt 10.4.2).

Während also ein UDDI-Verzeichnisdienst einen globalen »Wer-liefert-welchen-Web-Service« Katalog darstellt, bietet WS-Inspection eine strukturierte Methode, um auf Informationen über Dienste eines Anbieters zu verweisen.

10.4.1 Dienstverzeichnis UDDI

Das Ziel der UDDI-Initiative ist die Festlegung eines Standards für Verzeichnisdienste von Web Services. Aus konzeptueller Sicht definiert UDDI ein verteiltes Datenbanksystem zur Speicherung von Dienst-Metadaten basierend auf offenen Standards und Protokollen. Wesentliche Eigenschaften eines solchen Systems wie ein globales und einheitliches Datenschema, eine Anfragesprache, ein Autorisierungskonzept und eine Replikationsstrategie werden dazu in UDDI definiert. Ein Transaktionskonzept lässt UDDI jedoch vermissen. Da Änderungen von Daten nur relativ selten und von wenigen Berechtigten auf voneinander unabhängigen Datenbeständen durchgeführt werden und die breite Öffentlichkeit nur Leseberechtigung besitzt, fällt dies in der Praxis nicht ins Gewicht. Um eine globale Verfügbarkeit des Verzeichnisses zu gewährleisten, ist geplant, viele lokale Installationen von UDDI-Servern zu einem globalen Verbund zusammenzuschließen, dessen Daten weltweit periodisch abgeglichen werden. Dieser Verbund, UDDI-Wolke genannt, soll dem Anwender wie ein einzelner UDDI-Server erscheinen. Abbildung 10-4 zeigt diesen Verbund schematisch. Natürlich ist es auch möglich, eine lokale Installation unabhängig vom globalen Verbund zu betreiben, beispiels-

weise um Informationen über Dienste nur innerhalb eines Intranets zur Verfügung zu stellen.

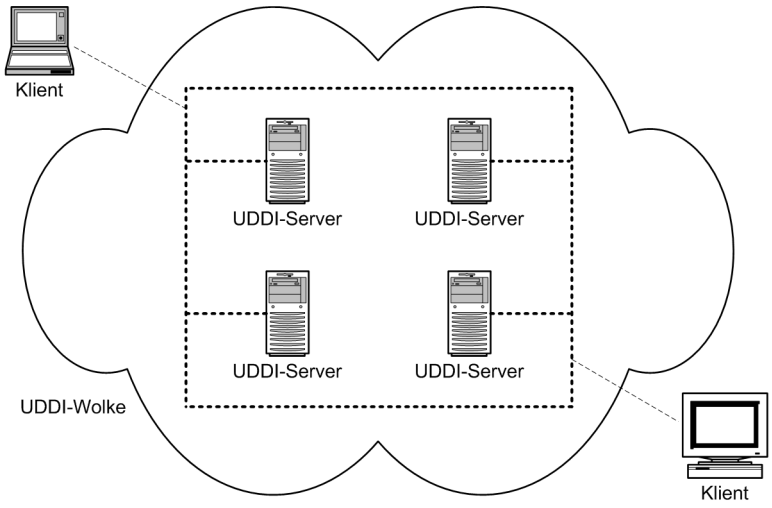


Abb. 10-4: *UDDI-Server-Verbund*

In den folgenden Abschnitten werden die wesentlichen Komponenten des »Datenbanksystems« UDDI vorgestellt. Nach einer Beschreibung des Datenschemas wird kurz die Anfragesprache vorgestellt und abschließend ein Überblick über das Replikationsverfahren gegeben.

Daten in UDDI

Eine wesentliche Voraussetzung für einen globalen Verzeichnisdienst ist die Spezifikation eines einheitlichen Schemas der zu speichernden (Meta-)Daten. Ein UDDI-Verzeichnisdienst unterscheidet konzeptionell drei verschiedene Klassen von Informationen:

- White Pages: Diese Klasse umfasst Daten über den Dienstanbieter (d. h. in der Regel eine Firma). Sie enthält neben Adressdaten und Daten über Kontaktpersonen eventuell auch weitere Identifikatoren von Unternehmen wie etwa Steuernummern oder ähnliches.
- Yellow Pages: Diese Klasse von Daten entspricht dem gedruckten Pendant – den gelben Seiten – insoweit, als sie verschiedene industrielle Kategorisierungen basierend auf Standard-Taxonomien umfasst (z. B. Universal Standard Products and Services Classification; UNSPSC). Im Gegensatz zur gedruckten Variante sind hier jedoch beliebig feine und vom Anbieter selbst definierte Kategorisierungen möglich.
- Green Pages: Zusätzlich zu den mehr administrativen Informationen der White Pages und Yellow Pages werden in einem UDDI-Verzeichnis auch tech-

nische Informationen über die angebotenen Dienste abgelegt. Dabei können sowohl Spezifikationsdokumente referenziert als auch Daten über konkrete Verfügbarkeitsorte, die so genannten Zugriffspunkte eines Dienstes, abgelegt werden. Details über die Spezifikationsdokumente, welche in der Regel WSDL-Dokumente sind, finden sich in Abschnitt 10.5.

Jede dieser Informationsklassen wird in einem UDDI-Verzeichnis auf festgelegte Datenstrukturen abgebildet. Ein wesentliches Prinzip des UDDI-Ansatzes ist die Verwendung von offenen und verbreiteten Standard-Internet-Protokollen. Deshalb werden alle Daten im XML-Format ausgetauscht und abgelegt. Als Kommunikationsprotokoll zwischen UDDI-Server und Client kommt SOAP zum Einsatz.

Datenstrukturen

Wie bereits erwähnt werden alle erfassten Daten auf das UDDI-XML-Schema abgebildet. Dieses umfasst fünf zentrale Datenstrukturen[5], deren Instanzen global eindeutig durch Universally Unique IDs (UUIDs) identifiziert werden.[6] Diese UUIDs entsprechen Identifikatoren in Datenbanksystemen. UDDI definiert die Strukturen *businessEntity* für Firmeninformationen, *businessService* für Klassen von angebotenen Diensten, *bindingTemplate* für technische Informationen, *tModel* (Abkürzung für »technical model«) zur Kategorienbildung und Referenzierung von technischen Informationen sowie *publisherAssertion* für die Modellierung von Geschäftsbeziehungen zwischen verschiedenen Firmen. Diese fünf Datenstrukturen zusammen mit kleineren Hilfsstrukturen sind in Abbildung 10-5 als UML-Klassendiagramm dargestellt.

Die zentralen Datenstrukturen businessEntity, businessService und bindingTemplate sind logisch in einer Baumstruktur angeordnet. Eine businessEntity kann mehrere registrierte businessServices haben, die in einer Eltern/Kind-Beziehung zur businessEntity stehen. Analog können einem businessService mehrere bindingTemplates zugeordnet sein. Die tModels stehen in keiner direkten Hierarchiebeziehung zu den anderen Datenstrukturen, sondern werden von diesen zu Klassifikationszwecken genutzt. Der Abschnitt »Einsatz von tModels« widmet sich dieser Datenstruktur im Detail. Durch publisherAssertions können Geschäftsbeziehungen wie etwa Allianzen, Partnerschaften oder Firmenbeziehungen wie Muttergesellschaft/Tochtergesellschaft beschrieben werden.

5. Aus Platzgründen kann hier nur ein kurzer Überblick über die Datenstrukturen gegeben werden. Für eine umfassende Darstellung sei auf [UDDI] verwiesen.
6. Die Struktur der UUIDs und der Erzeugungsalgorithmus sind im Standard ISO/IEC 11578:1996 beschrieben.

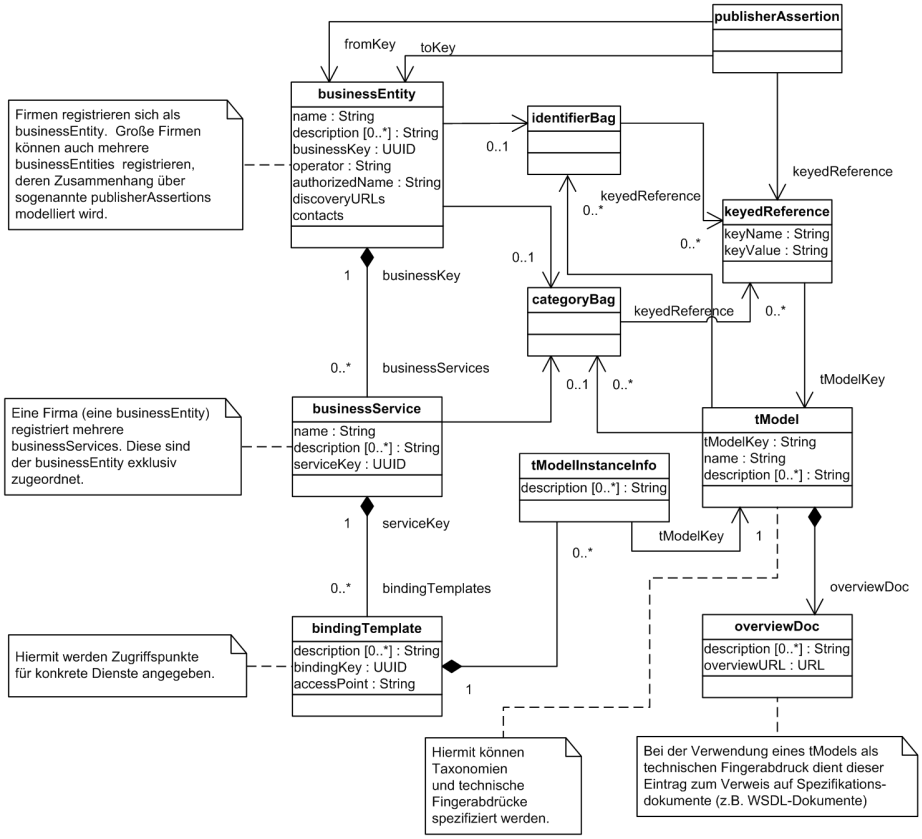


Abb. 10-5: *UML-Modell des UDDI-Schemas*

Wie sieht nun im praktischen Einsatz die Verwendung eines UDDI-Verzeichnisdienstes aus Sicht eines Dienstanbieters aus? Ein Anbieter registriert zunächst eine businessEntity. Dabei werden Angaben zu Namen, Adressen und Kontaktpersonen gemacht. Zusätzlich können verschiedene Taxonomien zur Identifikation (z. B. D-U-N-S[7] Nummern) sowie zur Kategorisierung des Eintrags verwendet werden (z. B. `ISO 3166` zur geographischen Kategorisierung). Die im Folgenden angegebenen XML-Dokumente können zur Registrierung bei UDDI-Servern verwendet werden (die konkreten SOAP-Nachrichten zur Registrierung folgen später), wobei eine Besonderheit zu beachten ist: Die hier aus Gründen der Vollständigkeit angegebenen UUIDs (`xxKey`-Attribute) dürfen bei der Registrierung nicht mit angegeben werden, da sie automatisch bei der Erstregistrierung durch den UDDI-Server vergeben werden. Bei späteren Änderungen dieser Daten bleiben jedoch die UUIDs unverändert, so dass sie zur Referenzierung der Elemente verwendet werden können.

7. Dun & Bradstreet Number (`www.dnb.com`): Weltweiter Identifikator für Unternehmen.

```
<?xml version="1.0" encoding="UTF-8" ?>
<businessEntity  businessKey="35C43D08-FBBF-2C59-AA29-CF032D054446">
  <name>Universitaet Passau - Lehrstuhl fuer Dialogorientierte Systeme</name>
  <description>Beispiel - Anwendung fuer Dienste rund um das Wetter</description>
  <categoryBag>
    <keyedReference   keyName="uddi-org:iso-ch:3166:1999" keyValue="DE-BY-PAS"
                      tModelKey="uuid:61668105-B6B6-425C-914B-409FB252C36D" />
  </categoryBag>
</businessEntity>
```

XML-Dokument 10-1: *businessEntity*

XML-Dokument 10-1 zeigt einen businessEntity-Eintrag, der für die Registrierung einer Organisation, hier Universitaet Passau, verwendet wurde. Der businessKey wurde bei der Registrierung automatisch vergeben. Als Kategorisierungsinformation ist angegeben, dass diese Organisation gemäß ISO-Klassifikation in Deutschland-Bayern-Passau (DE-BY-PAS) liegt. Kategorisierungsinformationen werden stets mit Hilfe von tModels innerhalb von keyedReference-Elementen mit den Attributen tModelKey, keyValue und keyName angegeben. Wie XML-Dokument 10-1 zeigt, werden keyedReferences eingesetzt, um Klassifizierungen mit Hilfe von tModels auszudrücken. Das verwendete tModel (angegeben im Attribut tModelKey) bezeichnet hier das abstrakte Konzept der geographischen ISO-Klassifizierung. Die Kategorie, der die businessEntity zugeordnet ist, steht im Attribut keyValue und muss im Rahmen der angegebenen Klassifizierung interpretiert werden.

Hat ein Anbieter eine businessEntity registriert, folgt als nächster Schritt die Angabe der Dienste. Die verschiedenen Arten von angebotenen Diensten werden unter verschiedenen businessServices zusammengefasst. In unserem Temperatur-Beispiel stellen der Temperaturdienst und der Einheitenumrechnerdienst zwei verschiedene businessServices dar. Im Gegensatz dazu sind der TemperaturInFahrenheitDienst und der Temperaturdienst dem gleichen businessService zugeordnet. Registriert der Dienstanbieter einen businessService, muss der businessKey der zugehörigen Firma angegeben werden, um eine eindeutige Zuordnung herstellen zu können. Das XML-Dokument 10-2 zeigt einen solchen businessService.

```
<?xml version="1.0" encoding="UTF-8" ?>
<businessService  serviceKey="362C66B3-03C4-F54B-AC4F-CDC8E2872EED"
                  businessKey="35C43D08-FBBF-2C59-AA29-CF032D054446">
<name>Temperaturauskunft</name>
<description>Der Dienst liefert die Temperatur an einem gegebenen Datum</description>
<categoryBag>
   <keyedReference keyName="uddi-org:iso-ch:3166:1999" keyValue="DE-BY-PAS"
                   tModelKey="uuid:61668105-B6B6-425C-914B-409FB252C36D" />
 </categoryBag>
</businessService>
```

XML-Dokument 10-2: *businessService*

Wie man sieht, stellt ein businessService lediglich eine Dienstgruppierung dar. Es wird noch keinerlei Auskunft über konkrete Dienste gegeben. Durch die Registrierung von businessEntity und businessService ist nun der Rahmen geschaffen, um konkrete Dienste registrieren zu können. Durch das Hinzufügen von Daten über konkrete Dienste mittels der Datenstruktur bindingTemplate werden Dienste an businessServices gebunden.

```
<?xml version="1.0" encoding="UTF-8"?>
<bindingTemplate      bindingKey="'4FB831F6-327E-7815-3331-52DB9BF9C220"
                      serviceKey="362C66B3-03C4-F54B-AC4F-CDC8E2872EED">
  <description>Temperaturangabe Passau in Grad Celsius</description>
  <accessPoint URLType="http">http://wetter.fmi.uni-passau.de:8004/service/Temperaturdienst</accessPoint>
  <tModelInstanceDetails>
    <tModelInstanceInfo tModelKey="uuid:B39997F5-DF7F-9F62-0EEE-6F43345DE1A3"/>
  </tModelInstanceDetails>
</bindingTemplate>
```

XML-Dokument 10-3: *bindingTemplate*

Das XML-Dokument 10-3 zeigt ein Dokument, das zur Registrierung des Temperaturdienstes verwendet werden kann. Der Dienst kann via HTTP über die angegebene URL angesprochen werden (URLType="http"). UDDI unterstützt neben HTTP noch andere Typen von Zugriffspunkten wie Fax, E-Mail etc. In einem bindingTemplate ist jedoch explizit noch keine Beschreibung konkreter Aufruf- und Rückgabemodalitäten (z. B. Signaturen) enthalten. Diese finden sich erst im tModel, welches durch den angegebenen tModelKey referenziert wird (siehe nächster Abschnitt).

Um Beziehungen zwischen Unternehmen zu beschreiben (z. B. Muttergesellschaft/Tochtergesellschaft, Allianzen, Partnerschaften), können publisherAssertions verwendet werden. Registriert eine Firma eine publisherAssertion, bleibt diese so lange für andere unsichtbar, bis der andere beteiligte Partner dieselbe publisherAssertion registriert. Erst dann wird diese gültig und öffentlich sichtbar. Durch diese symmetrische Registrierung kann Missbrauch weitestgehend verhindert werden.

Einsatz von tModels

Prinzipiell gibt es in UDDI zwei verschiedene Anwendungsszenarien für tModels:

1. tModel als technischer Fingerabdruck und
2. tModel als Namensraumbezeichner.

Dient ein tModel als technischer Fingerabdruck, wird es nur in der bindingTemplate-Datenstruktur verwendet. Dort referenziert es technische Dienstbeschreibungen, d. h., es verweist auf Spezifikationsdokumente. Als Namensraumbezeichner finden tModels unter anderem Verwendung in identifierBags, categoryBags und puplisherAssertions.

Technischer Fingerabdruck: Im XML-Dokument 10-3 wurde bereits gezeigt, dass sich in einem bindingTemplate ein Zugriffspunkt zu einem angebotenen Dienst befindet. Allerdings fehlten detaillierte Informationen für einen Anwendungsentwickler wie etwa Aufruf- und Rückgabeparameter, verwendete Transportprotokolle etc. Zur Beschreibung genau dieser Informationen wurde WSDL entwickelt, und die UDDI-Initiative hat deshalb eine Empfehlung herausgegeben (als so genanntes »Best practices«-Dokument [CuER01]), wie WSDL in UDDI eingesetzt werden kann, um diese Informationen anzugeben (siehe auch Abschnitt 10.5.3). Laut dieser Empfehlung sollen die Informationen nicht direkt in einer UDDI-Datenstruktur abgelegt werden, sondern in einem separaten WSDL-Dokument, das von einem tModel referenziert wird. Das in diesem Kapitel beschriebene Beispiel eines Temperaturdienstes folgt dieser Empfehlung.

```
<tModel tModelKey="uuid:B39997F5-DF7F-9F62-0EEE-6F43345DE1A3">
  <name>Temperaturauskunft</name>
  <description xml:lang="de">Die technische Spezifikation eines Temperaturdienstes</descrip-
tion>
  <overviewDoc>
    <description xml:lang="de">WSDL Dokument</description>
    <overviewURL>http://wetter.fmi.uni-passau.de/Temperaturdienst.wsdl</overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference  tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
                     keyName="uddi-org:types" keyValue="wsdlSpec"/>
  </categoryBag>
</tModel>
```

XML-Dokument 10-4: *tModel für Temperaturdienst*

Im XML-Dokument 10-4 fällt auf, dass die Art eines tModel (hier: tModel, das auf ein WSDL-Dokument verweist) durch ein generisches UDDI-Klassifikations-tModel (uuid:C1ACF…) angegeben wird. Es fällt ebenfalls auf, dass sich die UUIDs eines tModels von den UUIDs anderer UDDI-Datenstrukturen durch ein vorangestelltes uuid: unterscheiden.

Namensraumbezeichner: Die zweite Anwendungsmöglichkeit von tModels ist wesentlich allgemeiner als die eines technischen Fingerabdrucks. tModels können verwendet werden, um Taxonomien zu erzeugen und zu benutzen. Somit können eigene, anwendungsspezifische Klassifizierungen definiert und angewendet werden oder auch weltweit etablierte Standard-Klassifizierungen verwendet werden.

Beispiele für Standard-Klassifizierungen, die hauptsächlich bei businessEntities und businessServices Anwendung finden, sind: North American Industry Classification System (NAICS), Universal Standard Products and Services Classification (UNSPSC) oder ISO 3166 als internationaler Standard für geographische Regionen.

Geprüfte und ungeprüfte Taxonomien: Um eine global konsistente und korrekte Verwendung von Taxonomien mit zugehörigen Schlüssel/Wert-Paaren sicherzustellen, können in UDDI keyedReference-Einträge, die in identifierBags bzw. categoryBags auftreten, durch einen externen Validierungsdienst überprüft werden. Natürlich können ebenfalls Dokumente registriert werden, die beliebige, ungeprüfte Taxonomien verwenden. Insbesondere steht es jedem Anbieter frei, eigene tModels zur Erzeugung von Taxonomien zu definieren und diese dann ungeprüft verwenden zu lassen.

Anfragen

Nachdem in den vorangegangenen Abschnitten die Datenstrukturen vorgestellt wurden, die in einem UDDI-Verzeichnisdienst Verwendung finden, wird im Folgenden auf die zweite wesentliche Komponente des UDDI-»Datenbanksystems« eingegangen: die Anfragesprache. Generell finden alle Anfragen in einer XML-basierten Anfragesprache statt. Sowohl die Anfragen als auch deren Ergebnisse werden über das SOAP-Protokoll zwischen UDDI-Server und Client übertragen. UDDI unterstützt nur einen sehr kleinen Teil der vollen Funktionalität des SOAP-Protokolls, beispielsweise werden keine SOAP-Header-Elemente unterstützt.

Insgesamt definiert Version 2.0 von UDDI ein API mit etwa 25 Anfragen und 15 Antwortdokumenten.[8] Die UDDI-Spezifikation beschränkt sich bewusst auf relativ einfache Anfragetypen (beispielsweise sind keine Joins möglich) und überlässt es höheren Schichten wie etwa Suchmaschinen oder Marktplätzen, komplexere Anfragen durch Middleware-Funktionalität zu unterstützen. Die Anfragesprache von UDDI umfasst lediglich grundlegende Operationen zum Einfügen, Ändern, Löschen und Auffinden von Daten.

Prinzipiell lässt sich das API aufteilen in einen Teil zur Datenabfrage (Inquiry-API) und einen Teil zur Datenmodifikation (Publishing-API).

Inquiry-API: Innerhalb dieses API können Browse-Anfragen, die zum erstmaligen Auffinden von Metadaten dienen, und Drill-down-Anfragen, die vollständige und detaillierte Daten zurückliefern, unterschieden werden. Auf dieses API greift zum Beispiel ein UDDI-Browser zurück, der Clients eine interaktive Dienstsuche bietet.

Browse-Anfragen. Anfragen dieses Typs beginnen mit find_xx. Anhand der Anfrage aus XML-Dokument 10-5 werden die Möglichkeiten dieses Anfragetyps verdeutlicht.

```
<find_business maxRows="5" generic="2.0" xmlns="urn:uddiorg:api_v2">
  <findQualifiers>
    <findQualifier>sortByNameAsc</findQualifier>
  </findQualifiers>
  <name>Universitaet</name>
</find_business>
```

XML-Dokument 10-5: *Syntax einer find_business-Anfrage*

8. Im Rahmen dieses Abschnitts können nur einige Anfragebeispiele gezeigt werden. Für eine umfassende Referenz sei auf [UDDI] verwiesen.

Mit dieser Anfrage kann nach businessEntities gesucht werden, deren Name mit Universitaet beginnt.[9] Zusätzlich wurde spezifiziert, dass höchstens fünf Treffer zurückgeliefert werden (maxRows="5") und dass die Ergebnisse aufsteigend nach Namen sortiert sein sollen (sortByNameAsc). Als Ergebnis liefert diese Anfrage eine Liste von businessInfo-Elementen. Diese enthalten neben businessEntity-Daten auch Informationen über registrierte businessServices der gefundenen Firmen. Es existiert noch eine Reihe von weiteren möglichen Selektionskriterien. Beispielsweise kann eine Einschränkung auf Firmen erfolgen, die Dienste anbieten, die bestimmten tModels genügen.

Drill-down-Anfragen: Dieser Anfragetyp wird verwendet, wenn man einen Identifikator bereits kennt und die dazugehörigen Detaildaten abfragen will. Diese Anfragen beginnen mit get_xx. Als konkretes Beispiel dient die get_tModelDetail-Anfrage aus XML-Dokument 10-6. Sie liefert die Informationen über ein tModel eingebettet in ein tModelDetail-Element zurück.

```
<get_tModelDetail generic="2.0" xmlns="urn:uddi-org:api_v2">
  <tModelKey>uuid:B39997F5-DF7F-9F62-0EEE-6F43345DE1A3</tModelKey>
</get_tModelDetail>
```

XML-Dokument 10-6: *Syntax einer get_tModelDetail-Anfrage*

Publishing-API: Neben den Anfragen, durch die Metadaten abgefragt werden können, gibt es entsprechende Aufrufe, um Metadaten registrieren, verändern und löschen zu können. Diese beginnen mit save_xx zur Registrierung bzw. zur Änderung und mit delete_xx zum Löschen von Daten. Aus Sicherheitsgründen arbeiten alle Aufrufe dieses Publishing-API mit verschlüsselten Verbindungen zum UDDI-Server (via HTTPS mit SSL 3.0), und der Client muss sich durch ein Authentifizierungstoken ausweisen. Dieses Token erhält der Client durch Aufruf von get_authToken vom UDDI-Server. Der save_business-Aufruf aus XML-Dokument 10-7 kann zur Registrierung einer businessEntity verwendet werden.

```
<save_business generic="2.0" xmlns="urn:uddi-org:api_v2" >
  <authInfo>xy_token</authInfo>
  <businessEntity businessKey="">
    <name>Universitaet Passau - Lehrstuhl fuer Dialogorientierte Systeme</name>
    <description>Beispiel - Anwendung fuer Dienste rund um das Wetter</description>
    <!-- Hier können weitere Daten zu businessServices, bindingTemplates, etc. stehen -->
  </businessEntity>
</save_business>
```

XML-Dokument 10-7: *save_business-Aufruf*

9. Die UDDI-Spezifikation legt fest, dass ein Präfixvergleich durchgeführt wird, wenn bei der Suche keine Wildcards verwendet werden.

Neben dem authInfo-Element für das Authentifizierungstoken besitzt der Aufruf ein businessEntity-Element. Darin können die relevanten Informationen (siehe UML-Diagramm) registriert werden, wobei zu beachten ist, dass bei der Erstregistrierung ein leerer businessKey (businessKey="") übergeben werden muss. Der Schlüssel selbst wird vom UDDI-Server generiert. Gleichzeitig können hierbei als Unterelemente des businessEntity-Eintrags auch businessServices und bindingTemplates registriert werden. Auch bei diesen bedeutet die Übergabe eines leeren Schlüssels die Neuregistrierung. Werden stattdessen bestehende Schlüssel verwendet, werden dadurch bestehende Einträge geändert.

Pufferung und Puffer-Kohärenz: Die UDDI-Initiative schlägt für den Umgang mit bindingTemplate-Informationen ein spezielles Vorgehen vor. Dieses Vorgehen wird anhand des folgenden Beispiels beschrieben: Eine Anwendung A, welche den Temperaturdienst T benutzen will (die UUID von T muss bereits zur Entwicklungszeit von A bekannt sein), geht folgendermaßen vor:

1. Um T erstmalig aufrufen zu können, müssen die Metadaten von T mit get_bindingTemplate vom UDDI-Server abgerufen werden. Die dabei erhaltenen technischen Informationen (insbesondere der Zugriffspunkt) können von A lokal gepuffert werden, um die Belastung des UDDI-Servers gering zu halten, beispielsweise wenn A den Dienst T mehrfach benötigt.
2. Schlägt ein Aufruf von T fehl (tritt beispielsweise ein HTTP-404-Fehler auf), muss A erneut mittels einer get_bindingDetail-Anfrage die (hoffentlich) aktualisierten Metadaten von T vom UDDI-Server abrufen. Unterscheiden sich diese von der gepufferten (alten) Version, sollen die neuen Metadaten gepuffert und die alte Version aus dem lokalen Cache entfernt werden.

Damit will die UDDI-Initiative einerseits einer Überlastung der UDDI-Server vorbeugen, indem es eine lokale Pufferung der Anfrageergebnisse empfiehlt, andererseits aber einer Überalterung und damit einer Unbrauchbarkeit der gepufferten Daten vorbeugen, indem im Falle eines Fehlers eine erneute Anfrage beim UDDI-Server erfolgt. Aus Datenbanksicht beschreibt dieses Vorgehen ein Verfahren, um Puffer-Kohärenz zu erzielen.

Replikation

Nachdem bereits die Datenstrukturen und die Anfragesprache des UDDI-»Datenbanksystems« vorgestellt wurden, soll hier kurz auf das Replikationsverfahren in UDDI eingegangen werden. Die Motivation für die Definition eines globalen, verteilten Verzeichnisdienstes ist offensichtlich: Die Verwendung von UDDI ist umso attraktiver, je größer die Datenbasis des UDDI-Servers ist. Umfasst diese beispielsweise nur deutsche Dienste, entgehen dem Benutzer unter Umständen alternative Dienste aus Europa. Aus diesem Grund wird ein weltweiter Verzeichnisdienst (UDDI-Wolke) angestrebt, der verschiedene Instanzen von UDDI-Servern besitzt, dessen Gesamtdaten sich jedoch über jede Instanz abfragen lassen. Der Datenabgleich zwischen den Servern soll durch eine Replikationsrichtlinie erreicht wer-

den. In der Datenbankliteratur [Dada96] findet sich eine Vielzahl von Ansätzen zur Replikation und Allokation von Daten. Speziell für Metadaten-Server-Verbunde wurden auch Ansätze entwickelt, die auf Publish-Subscribe-Architekturen beruhen und versuchen, den erforderlichen Kommunikationsaufwand zu reduzieren, z. B. [KKKK01, KKKK02].

Die UDDI-Initiative schlägt zum Datenabgleich zwischen den UDDI-Servern einen Primärkopie-Ansatz vor, der darauf beruht, dass jedes Datum im Verzeichnis einen Heimat-UDDI-Server besitzt, an dem es registriert wurde und an dem es geändert bzw. gelöscht werden kann. Die Daten werden gewissermaßen an diesem Heimat-Server verwahrt. Periodisch werden die anderen UDDI-Server benachrichtigt und ihnen wird der aktuelle Replikationsstatus des meldenden Servers mitgeteilt. Stellt ein Server fest, dass er veraltete Daten besitzt, weil seit dem letzten Abgleich auf einem anderen Server Änderungen vorgenommen wurden, fordert er die entsprechenden Änderungsdatensätze an. Änderungen werden dabei mit Hilfe von global eindeutigen, aufsteigenden Sequenznummern erkannt. Die gesamte zur Replikation erforderliche Kommunikation erfolgt via SOAP auf verschlüsselten HTTP-Verbindungen.

Die in den vorangegangenen Abschnitten beschriebenen Strukturen und Eigenschaften von UDDI beziehen sich auf Version 2 von UDDI. Die Version 3 des UDDI-Standards wird Spezifikationen zur Authentifizierung, zur Autorisierung, zur Versionskontrolle und zur Historisierung von Änderungen enthalten.

10.4.2 Dienstsuche: WS-Inspection

Wie bereits erwähnt ist ein UDDI-Verzeichnis nicht die einzige Möglichkeit, Dienste auffindbar zu machen. Ein Ansatz, der nicht als Konkurrenz, sondern als Ergänzung zu UDDI und anderen Verfahren gedacht ist, ist die Web Services Inspection Language (WS-Inspection). Ein WS-Inspection-Dokument ist im Wesentlichen eine Sammlung von Verweisen auf bereits existierende Dokumente, die Web Services beschreiben. Damit bietet WS-Inspection die Möglichkeit, eine Sammlung aller Referenzen auf Beschreibungen von Diensten dort verfügbar zu machen, wo die Dienste auch angeboten werden, ohne dass die Beschreibungen repliziert werden müssen. Ein WS-Inspection-Dokument kann prinzipiell Verweise auf beliebige Dokumente beinhalten, sieht allerdings spezielle Elemente für Verweise auf Dokumente, die in UDDI-Verzeichnissen gespeichert sind, sowie für WSDL-Dokumente vor und ist durch seinen Erweiterungsmechanismus flexibel erweiterbar. Wir werden uns in diesem Kapitel allerdings auf die wichtigsten Möglichkeiten von WS-Inspection in Bezug auf UDDI und WSDL konzentrieren.

```
<?xml version="1.0" encoding="UTF-8"?>
<inspection   xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/"
              xmlns:wsiluddi="http://schemas.xmlsoap.org/ws/2001/10/inspection/uddiv2/">
  <service>
    <abstract>Der Dienst liefert die Temperatur an einem gegebenen Datum.</abstract>
    <name>Temperaturdienst</name>
    <description    referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
                    location="http://wetter.fmi.uni-passau.de/Temperaturdienst.wsdl" />
    <description referencedNamespace="urn:uddi-org:api_v2">
      <wsiluddi:serviceDescription location="http://wetter.fmi.uni-passau.de:9004/uddi/inquiryapi/uddi">
        <wsiluddi:serviceKey>362C66B3-03C4-F54B-AC4F-CDC8E2872EED</wsiluddi:serviceKey>
      </wsiluddi:serviceDescription>
    </description>
  </service>

  <service>
    <abstract>Der Dienst rechnet Temperaturangaben in verschiedene Einheiten um.</abstract>
    <name>Einheitenumrechnerdienst</name>
    <description   referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
                   location="http://wetter.fmi.uni-passau.de/Einheitenumrechnerdienst.wsdl" />
    <description referencedNamespace="urn:uddi-org:api_v2">
      <wsiluddi:serviceDescription location="http://wetter.fmi.uni-passau.de:9004/uddi/inquiryapi/uddi">
        <wsiluddi:serviceKey>2C78DBF9-4295-FBAC-7CC7-62F9609AB865</wsiluddi:serviceKey>
      </wsiluddi:serviceDescription>
    </description>
  </service>

  <service>
    <abstract>Der Dienst liefert die Temp. an einem gegebenen Datum in Grad Fahrenheit.</abstract>
    <name>TemperaturInFahrenheitDienst</name>
    <description   referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
                   location="http://wetter.fmi.uni-passau.de/TemperaturInFahrenheitDienst.wsdl" />
    <description referencedNamespace="urn:uddi-org:api_v2">
      <wsiluddi:serviceDescription location="http://wetter.fmi.uni-passau.de:9004/uddi/inquiryapi/uddi">
        <wsiluddi:serviceKey>362C66B3-03C4-F54B-AC4F-CDC8E2872EED</wsiluddi:serviceKey>
      </wsiluddi:serviceDescription>
    </description>
  </service>
</inspection>
```

WS-Inspection-Dokument 10-1: *Dienste von wetter.fmi.uni-passau.de*

WS-Inspection-Dokument 10-1 zeigt die Dienste eines Anbieters am Beispiel unseres Servers `wetter.fmi.uni-passau.de`. Dieser Server bietet drei Dienste an: den Temperaturdienst, den Einheitenumrechnerdienst und den TemperaturInFahrenheitDienst. Zu allen drei Diensten liegen sowohl entsprechende WSDL-Dokumente als auch Einträge in einem UDDI-Verzeichnisdienst vor. Ein WS-Inspection-Dokument hat einen sehr einfachen Aufbau: Das Wurzelelement heißt inspection und beinhaltet Verweise auf Dienstbeschreibungen oder Verweise auf andere WS-Inspection-Dokumente als Unterelemente. Für jeden Dienst, der in dem WS-Inspection-Dokument aufgeführt ist, existiert ein eigenes service-Element. Wir wer-

den die Bedeutung der verschiedenen Unterelemente eines service-Elementes am Beispiel des Temperaturdienstes beschreiben. Das erste Element im Beispiel ist das abstract-Element. Dieses Element ist optional, beinhaltet eine kurze textuelle Beschreibung des Dienstes und ist, genau wie das nachfolgende name-Element, für menschliche Leser und nicht zur maschinellen Auswertung gedacht. Das optionale name-Element gibt den Namen des Dienstes an. Danach folgen ein oder mehrere description-Elemente, die jeweils einen Verweis auf eine Beschreibung des Dienstes beinhalten. In dem Beispieldokument sind das jeweils ein Verweis auf UDDI und ein WSDL-Dokument. Zuerst erklären wir den Verweis auf das WSDL-Dokument:

```
<description  referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
              location="http://wetter.fmi.uni-passau.de/Temperaturdienst.wsdl" />
```

Das Attribut referencedNamespace bezeichnet den Namensraum, zu dem das referenzierte Dokument gehört. In unserem Fall handelt es sich um ein WSDL-Dokument. Dieser Namensraumbezeichner kann von einem Benutzer des WS-Inspection-Dokumentes genutzt werden, um festzustellen, ob er mit dem angegebenen Dokument umgehen kann und es deswegen wert ist geladen zu werden oder nicht. Das Attribut location spezifiziert eine URL, die genutzt werden kann, um das referenzierte Dokument zu laden. Dieses Attribut muss nicht angegeben werden, wenn man den Erweiterungsmechanismus von WS-Inspection nutzt. Für unser Beispiel reicht allerdings dieser Standardansatz aus, um auf das WSDL-Dokument zu verweisen. Die Referenz auf eine Beschreibung, die in einem UDDI-Verzeichnisdienst gespeichert ist, demonstriert den Einsatz des Erweiterungsmechanismus am Beispiel der standardisierten UDDI-Erweiterung. Hier legt das Attribut referencedNamespace fest, dass die Beschreibung in einem UDDI-Verzeichnis liegt, das die Protokollversion 2 versteht. Wie man zu der Beschreibung gelangt, ist in diesem Fall durch Unterelemente festgelegt: Das location-Attribut des serviceDescription-Elementes spezifiziert die URL, unter der das Inquiry-API des UDDI-Verzeichnisdienstes angesprochen werden kann. Der Wert des serviceKey-Elementes kann dann dazu verwendet werden, um an den UDDI-Verzeichnisdienst eine get_serviceDetail-Nachricht via SOAP zu schicken und die gewünschten Informationen zu erhalten.

Die Möglichkeiten von WS-Inspection gehen über die hier gezeigten hinaus. Zum einen ist WS-Inspection flexibel erweiterbar, zum anderen erlaubt es, Hierarchien von WS-Inspection-Dokumenten aufzubauen. Diese Möglichkeit ist für größere Firmen mit mehreren Abteilungen sehr wichtig, da damit ein zentrales WS-Inspection-Dokument der Firma nur auf die Dokumente der Abteilungen verweisen muss und die Abteilungen ihre eigenen WS-Inspection-Dokumente selbst verwalten können. Außerdem kann innerhalb von WS-Inspection-Dokumenten auch auf businessEntity-Einträge in UDDI-Verzeichnisdiensten verwiesen werden, statt auf einzelne businessService-Einträge wie im Beispiel.

Nachdem wir das Format von WS-Inspection-Dokumenten vorgestellt haben, bleibt noch die Frage zu klären, wie WS-Inspection-Dokumente gefunden werden können. Dazu werden im Standard zwei Möglichkeiten festgelegt: Zum einen

wird vorgeschlagen, WS-Inspection-Dokumente unter dem Namen »inspection.wsil« an den Hauptseiten des Web-Servers des Anbieters abzulegen. In unserem Fall ist das WS-Inspection-Dokument unseres Servers also unter der Adresse `http://wetter.fmi.uni-passau.de/inspection.wsil` abrufbar. Die zweite Möglichkeit besteht darin, von HTML-Seiten aus mit Hilfe eines META-Elementes auf WS-Inspection-Dokumente zu verweisen. Diese Möglichkeit wird im HTML-Dokument 10.1 demonstriert. Ein geeigneter Browser kann den Benutzer dann auf vorhandene WS-Inspection-Dokumente hinweisen.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//DE">
<HTML>
  <HEAD>
    <TITLE>Web-Dienste am Beispiel eines Temperaturdienstes</TITLE>
    <META name="serviceInspection" content="http://wetter.fmi.uni-passau.de/inspection.wsil">
  </HEAD>

  <BODY>
    <!-- ... Text der HTML-Seite ... -->
  </BODY>
</HTML>
```

HTML-Dokument 10-1: *Verweis auf ein WS-Inspection-Dokument in einer HTML-Seite*

10.5 Dienstbeschreibung (WSDL)

Das Finden eines Dienstes mit Hilfe von UDDI oder WS-Inspection reicht nicht aus, um diesen Dienst auch aufrufen zu können. Dazu benötigt man die genauen technischen Spezifikationen, die festlegen, welche Operationen der Dienst unterstützt, wie das Format der Eingabe- und Ausgabedaten beschaffen sein muss und wie die Übertragung der Daten zu geschehen hat, d. h. über welches Protokoll mit dem Dienst kommuniziert wird. Genau dafür wurde die Web Services Description Language (WSDL) [CCMW01] von Ariba, IBM und Microsoft entwickelt und beim W3C zur Standardisierung eingereicht.

Die WSDL-Spezifikation legt fest, auf welche Weise man die zum Aufruf eines Dienstes notwendigen Informationen in einem XML-Dokument ablegt. Im Folgenden werden wir die einzelnen Komponenten eines WSDL-Dokuments am Beispiel des Temperaturdienstes vorstellen.

10.5.1 Struktur eines WSDL-Dokuments

WSDL verwendet zur Beschreibung von Diensten sechs verschiedene Elemente: types, message, portType, binding, port und service (siehe Abbildung 10.66). WSDL betrachtet einen Dienst (service) als eine Menge von (abstrakten) Endpunkten von Netzwerkverbindungen. Diese Endpunkte tauschen Nachrichten (messages) untereinander aus, wobei eine Nachricht eine abstrakte Beschreibung der ausgetauschten Daten darstellt. Operationen repräsentieren abstrakte Beschreibungen

möglicher Aktionen eines Dienstes und bestehen prinzipiell aus Ein- und Ausgabenachrichten. Operationen werden zu Port-Typen (portTypes) zusammengefasst. Dabei handelt es sich im Grunde um eine Menge von Operationen, die von einem oder mehreren Endpunkten unterstützt werden. Bis hierhin sind die jeweiligen Definitionen unabhängig von bestimmten Netzwerkprotokollen und Datenformaten. Erst eine so genannte Bindung (binding) legt in WSDL für einen Port-Typ ein bestimmtes Netzwerkprotokoll und Datenformat fest. Ein Port (port) fasst anschließend eine Netzwerkadresse mit einer Bindung zusammen. Ein Dienst selbst besteht dann aus einer Menge von (zusammengehörigen) Ports.

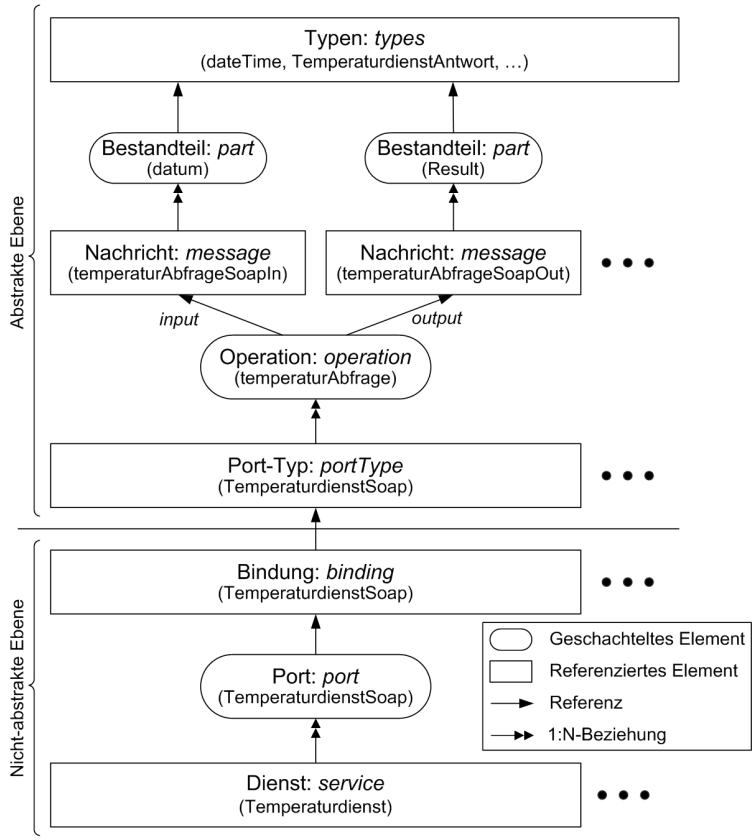


Abb. 10-6: *Struktureller Aufbau eines WSDL-Dokumentes*

Ein WSDL-Dokument besteht prinzipiell aus einer Menge von Definitionen. Jede Definition wird durch ein eigenes Element im WSDL-Dokument repräsentiert. Jedes dieser Elemente muss ein Unterelement des Wurzelelements definitions sein. Als vollständiges Beispiel zeigt WSDL-Dokument 10-1, wie der Temperaturdienst in WSDL beschrieben wird. Abbildung 10-6 zeigt den strukturellen Aufbau eines WSDL-Dokuments am Beispiel des Temperaturdienstes aus WSDL-Dokument 10-1.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions   name="Temperaturdienst"
               targetNamespace="http://wetter.fmi.uni-passau.de/Temperaturdienst.wsdl"
               xmlns:tns="http://wetter.fmi.uni-passau.de/Temperaturdienst.wsdl"
               xmlns:ns1="http://wetter.fmi.uni-passau.de/Temperaturdienst.xsd"
               xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <schema  xmlns="http://www.w3.org/2001/XMLSchema"
             targetNamespace="http://wetter.fmi.uni-passau.de/Temperaturdienst.xsd">
      <complexType name="TemperaturdienstAntwort">
        <sequence>
          <element name="temperatur" type="double"/>
          <element name="einheit" type="string"/>
          <element name="messdatum" type="dateTime"/>
        </sequence>
      </complexType>
    </schema>
  </types>
  <message name="temperaturAbfrageSoapIn">
    <part name="datum" type="xsd:dateTime"/>
  </message>
  <message name="temperaturAbfrageSoapOut">
    <part name="Result" type="ns1:TemperaturdienstAntwort"/>
  </message>
  <portType name="TemperaturdienstSoap">
    <operation name="temperaturAbfrage" parameterOrder="datum">
      <input name="temperaturAbfrageSoapIn" message="tns:temperaturAbfrageSoapIn"/>
      <output name="temperaturAbfrageSoapOut" message="tns:temperaturAbfrageSoapOut"/>
    </operation>
  </portType>
  <binding name="TemperaturdienstSoap" type="tns:TemperaturdienstSoap">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="temperaturAbfrage">
      <soap:operation soapAction="temperaturAbfrage" style="rpc"/>
      <input name="temperaturAbfrageSoapIn">
        <soap:body   use="encoded" namespace="http://wetter.fmi.uni-passau.de/Temperaturdienst"
                     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output name="temperaturAbfrageSoapOut">
        <soap:body   use="encoded" namespace="http://wetter.fmi.uni-passau.de/Temperaturdienst"
                     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>
  <service name="Temperaturdienst">
    <port name="TemperaturdienstSoap" binding="tns:TemperaturdienstSoap">
      <soap:address location="http://wetter.fmi.uni-passau.de:8004/service/Temperaturdienst"/>
    </port>
  </service>
</definitions>
```

WSDL-Dokument 10-1: *Beschreibung des Temperaturdienstes*

WSDL erlaubt es, die Beschreibung von Diensten zu modularisieren. Dazu lagert man einzelne, zusammengehörende Teile in eigene Dateien aus und importiert diese dann mit einem import-Element in das Hauptdokument:[10]

```
<definitions ...>
  <import namespace="uri" location="uri"/> *
</definitions>
```

Zum Beispiel kann man die Definition der Datentypen von Nachrichten in eine eigene Datei auslagern und mit Hilfe eines solchen Elements in das WSDL-Dokument einbinden.

Bei der Definition des WSDL-Standards wurde darauf geachtet, Erweiterbarkeit zu gewährleisten. Dies ist insbesondere bei der Bindung von abstrakten Diensten an bestimmte Protokolle und Datenformate wichtig. WSDL erlaubt die Angabe von so genannten Erweiterungselementen als Unterelemente von bestimmten WSDL-Elementen. Mit ihnen können etwa technische Spezifikationen für eine Bindung angegeben werden, da diese im Allgemeinen vom verwendeten Protokoll und Datenformat abhängen.

Im Folgenden werden nun die einzelnen Definitionen, d. h. die Unterelemente des definitions-Wurzelelements eines WSDL-Dokuments, beschrieben. Zuerst werden die Elemente beschrieben, die zur Beschreibung der abstrakten Teile von Diensten verwendet werden. Die Beschreibung der Elemente für die nicht abstrakten Teile folgt in Abschnitt 10.5.2.

Typen

Das types-Element dient zur Definition von Datentypen, die in den ausgetauschten Nachrichten verwendet werden. Die Syntax sieht folgendermaßen aus:

```
<definitions ...>
  <types>
    <xsd:schema ... /> *
  </types>
</definitions>
```

Aus Gründen der Interoperabilität wird die Verwendung von XML Schema zur Definition von Typen bevorzugt. Mit Hilfe von Erweiterungselementen können aber auch andere Typ-Systeme eingebunden werden. Statt die Typen im WSDL-Dokument zu definieren, kann man auch bereits existierende XML-Schema-Dokumente einbinden (mit Hilfe von import) und die darin definierten Datentypen verwenden.

10. Zur Beschreibung der XML-Grammatik wird die gleiche informelle Syntax benutzt wie in der WSDL-Spezifikation. Ein * hinter einem Element bedeutet, dass es beliebig oft (auch 0 mal) vorkommen kann, ein ?, dass es 0 oder 1 mal vorkommen kann.

Nachrichten

Nachrichten dienen in WSDL dazu, das Format der Ein- und Ausgabedaten von Diensten festzulegen. Jede Nachricht wird in einem message-Element definiert. Die Syntax dazu sieht folgendermaßen aus:

```
<definitions ...>
  <message name="nmtoken"> *
    <part name="nmtoken" element="qname"? type="qname"? /> *
  </message>
</definitions>
```

Ein WSDL-Dokument kann beliebig viele Nachrichten (und damit message-Elemente) enthalten. Jede Nachricht besteht aus einem oder mehreren logischen Bestandteilen, die jeweils in einem part-Element definiert werden. Wenn man Nachrichten mit Funktionen vergleicht, dann entsprechen die Bestandteile einer Nachricht den Funktionsparametern. Der Name einer Nachricht (Attribut name) muss eindeutig innerhalb aller Nachrichten des Dokuments sein. Der Name eines Bestandteils muss eindeutig innerhalb aller Bestandteile der umschließenden Nachricht sein. Jedem Bestandteil wird mit Hilfe von speziellen Attributen ein Typ (aus einem Typsystem) zugeordnet. Für die Verwendung von XML-Schema-Typen bietet WSDL standardmäßig die Attribute element und type an. Die Typbeschreibung einer Nachricht gibt deren Inhalt nur abstrakt wieder, d. h., das tatsächliche Format zur Übertragung der Nachricht kann – abhängig von der verwendeten Bindung – durchaus davon abweichen.

Im WSDL-Dokument 10-1 werden die beiden Nachrichten temperaturAbfrageSoapIn und temperaturAbfrageSoapOut definiert. Beide bestehen aus je einem Bestandteil. Während der Bestandteil datum der Nachricht temperaturAbfrageSoapIn mit dateTime einen Typ besitzt, der bereits in XML-Schema definiert ist, ist der Typ des Bestandteils Result benutzerdefiniert und stammt aus dem types-Element.

Port-Typen

WSDL verwendet Port-Typen, um eine Menge von abstrakten Operationen und die dazugehörigen abstrakten Nachrichten unter einem gemeinsamen Namen zusammenzufassen. Die Syntax dazu sieht folgendermaßen aus:

```
<definitions ...>
  <portType name="nmtoken"> *
    <operation name="nmtoken" ... /> *
  </portType>
</definitions>
```

Der Name eines Port-Typs muss eindeutig innerhalb aller Port-Typen eines WSDL-Dokuments sein. Jede Operation wird in einem eigenen operation-Element definiert. Der Name einer Operation muss eindeutig innerhalb des umschließen-

den Port-Typs sein. Grundsätzlich stehen vier Operationsprimitiven zur Verfügung, die definieren, welche Nachrichten und wie Nachrichten empfangen und gesendet werden:

- One-Way: Der Endpunkt empfängt eine Nachricht
- Request-Response: Der Endpunkt empfängt eine Nachricht und sendet eine korrelierte Nachricht als Antwort.
- Solicit-Response: Der Endpunkt sendet eine Nachricht und empfängt eine korrelierte Nachricht als Antwort.
- Notification: Der Endpunkt sendet eine Nachricht.

Ein input-Unterelement eines operation-Elements definiert das abstrakte Nachrichtenformat für eine zu empfangende Nachricht. Ein output-Unterelement definiert das Format der zu sendenden Nachricht. Ein fault-Unterelement definiert mögliche auftretende Fehlernachrichten. Art und Reihenfolge der Unterelemente eines operation-Elements bestimmen den Typ einer Operation. Operationen vom Typ Request-Response besitzen ein input, ein output- und ein oder mehrere fault-Unterelemente (in genau dieser Reihenfolge). Solicit-Response-Operationen können ebenfalls diese unterschiedlichen Unterelemente haben, allerdings in der Reihenfolge output-Unterelement, input-Unterelement und fault-Unterelemente. Request-Response- und Solicit-Response-Operationen unterscheiden sich also nur in der Reihenfolge, in der ihre Unterelemente definiert werden. One-Way-Operationen besitzen nur ein einzelnes input-Unterelement, Notification-Operationen nur ein einzelnes output-Unterelement. Jedem der Unterelemente wird mit Hilfe eines message-Attributs eine Nachricht zugeordnet. Der Name jedes Unterelements muss eindeutig innerhalb des umschließenden Port-Typs sein. Wird einem der Elemente kein Name zugewiesen, ordnet WSDL automatisch einen Namen zu. Dieser entspricht dem Namen der umschließenden Operation im Falle von One-Way- oder Notification-Operationen. Im Falle von Request-Response- und Solicit-Response-Operationen entspricht der Name dem Namen der Operation mit einem angehängten »Request« bzw. »Solicit« für die Eingangsnachricht und einem angehängten »Response« für die Ausgangsnachricht.

Folgender Auszug aus dem WSDL-Dokument 10-1 definiert die Operation temperaturAbfrage:

```
<operation name="temperaturAbfrage" parameterOrder="datum">
  <input name="temperaturAbfrageSoapIn" message="tns:temperaturAbfrageSoapIn"/>
  <output name="temperaturAbfrageSoapOut" message="tns:temperaturAbfrageSoapOut"/>
</operation>
```

Diese Operation ist vom Typ Request-Response, da zuerst ein input-Element und dann ein output-Element definiert wird (fault-Elemente wurden keine definiert). Die Dokumente SOAP-Dokument 10-1 und SOAP-Dokument 10-2, die bereits in Abschnitt 10.3 gezeigt wurden, sind Beispiele für Ein- und Ausgabenachrichten dieser Operation. Wird eine Nachricht später zusammen mit einer RPC-Bindung

verwendet, besteht bei Request-Response- und Solicit-Response-Operationen die Möglichkeit, die Reihenfolge der Parameter anzugeben, wie sie in der Signatur der ursprünglichen RPC-Funktion festgelegt ist. Dazu dient das Attribut parameterOrder. Der Wert dieses Attributs ist im Allgemeinen eine Liste von Namen von Nachrichtenbestandteilen, jeweils durch ein Leerzeichen getrennt.

10.5.2 Verknüpfung mit Kommunikationsprotokollen und Nachrichtenformaten

Mit den bisherigen Definitionen wurden Typen, abstrakte Nachrichten, Operationen und Port-Typen definiert. Die nun folgenden Definitionen für Bindungen, Ports und Dienste binden diese abstrakten Definitionen an tatsächliche Nachrichtenformate, Protokolle und Netzwerkadressen.

Bindungen

Eine Bindung legt für einen bestimmten Port-Typ ein spezifisches Protokoll und Nachrichtenformat für die Übertragung fest. Einem Port-Typ können mehrere unterschiedliche Bindungen zugeordnet werden, z. B. Bindungen für SOAP, für HTTP und für MIME. Die Syntax einer Bindung sieht folgendermaßen aus:

```
<definitions .... >
  <binding name="nmtoken" type="qname"> *
    <!-- Erweiterungselemente -->
    <operation name="nmtoken"> *
      <!-- Erweiterungselemente -->
      <input name="nmtoken"? > ?
        <!-- Erweiterungselemente -->
      </input>
      <output name="nmtoken"? > ?
        <!-- Erweiterungselemente -->
      </output>
      <fault name="nmtoken"> *
        <!-- Erweiterungselemente -->
      </fault>
    </operation>
  </binding>
</definitions>
```

Zur Definition einer Bindung wird das binding-Element verwendet. Der Name der Bindung muss eindeutig innerhalb aller Bindungen eines WSDL-Dokuments sein. Das type-Attribut legt den Port-Typ fest, für den die Bindung definiert wird. Die tatsächlichen Bindungsinformationen werden mit Hilfe von Erweiterungselementen definiert, da die Art der Informationen und damit die Menge (Namen und Anzahl) der Elemente von der Art der Bindung abhängig sind. WSDL ermöglicht die Definition von Bindungsinformationen mit Hilfe von Erweiterungselementen für zu empfangende Nachrichten (im input-Element), zu sendende Nachrichten (im output-Element) und Fehlernachrichten (in den fault-Elementen). Außerdem kön-

nen Erweiterungselemente benutzt werden, um Bindungsinformationen für spezifische Operationen (in den operation-Elementen) sowie ganz allgemein für die Bindung (im binding-Element) anzugeben.

Das WSDL-Dokument 10-1 enthält eine Bindung für das SOAP-Protokoll mit Namen TemperaturdienstSoap. Protokollspezifische Informationen werden mit Hilfe der Erweiterungselemente soap:binding, soap:operation und soap:body festgelegt.

Ports und Dienste

Ein Port definiert einen individuellen Endpunkt im Netz, indem er einer Bindung eine einzelne Netzwerkadresse zuordnet. Ein Dienst gruppiert schließlich mehrere zusammengehörige Ports. Ports werden in einem port-Element definiert, Dienste in einem service-Element. Die Syntax dazu sieht folgendermaßen aus:

```
<definitions ... >
  <service name="nmtoken"> *
    <port name="nmtoken" binding="qname"> *
      <!-- Erweiterungselemente -->
    </port>
  </service>
</definitions>
```

Der Name eines Ports muss eindeutig innerhalb aller Ports eines WSDL-Dokuments sein. Der Name eines Dienstes muss eindeutig innerhalb aller Dienste eines WSDL-Dokuments sein. Das binding-Attribut eines Ports legt die Bindung fest, für die ein Port definiert wird. Die Zuordnung der Netzwerkadresse geschieht mit Hilfe von Erweiterungselementen. Das WSDL-Dokument 10-1 beispielsweise enthält eine Bindung an SOAP. Mit Hilfe des Erweiterungselements soap:address wird die SOAP-Adresse des Temperaturdienstes festgelegt, im Beispiel auf die URL `http://wetter.fmi.uni-passau.de:8004/service/Temperaturdienst`.

Neben der Bindung an das SOAP-Protokoll, die in dem Beispieldokument verwendet wird, beschreibt die Spezifikation von WSDL außerdem noch Bindungen an das HTTP-Protokoll und an das MIME-Protokoll.

Ports, die zu einem Dienst zusammengefasst werden, sollten nicht miteinander kommunizieren. Außerdem geht WSDL davon aus, dass Ports, die auf denselben Port-Typ verweisen, als Alternativen anzusehen sind, d. h. dass ihre Funktionsweise semantisch gesehen gleich ist.

10.5.3 Einbettung von WSDL in UDDI

Ein WSDL-Dokument dient dazu, Schnittstellen und Bindungen an Protokolle und Datenformate für Dienste festzulegen. Diese Informationen werden zum Teil auch für die Beschreibung eines Dienstes in UDDI benötigt. Die entsprechenden

Standardisierungsgremien haben sich deshalb auf einen Vorschlag geeinigt, um WSDL-Dokumente in UDDI zu verwenden [CuER01].

Dabei gehen die Standardisierungsgremien davon aus, dass entsprechende Gremien aus Industrie, Forschung und Politik eine Menge von Diensttypen festlegen und diese mit Hilfe von WSDL-Dokumenten beschreiben. Diese Dokumente enthalten die Beschreibung der Schnittstellen der Dienste sowie Bindungen für verschiedene Protokolle, nicht aber Port- und Dienst-Definitionen. Die WSDL-Dokumente werden als UDDI-tModels registriert. Das overviewDoc-Element eines solchen tModels verweist dabei auf das entsprechende WSDL-Dokument (mit Hilfe einer URL). Dienste, die das entsprechende tModel implementieren, müssen dann eine Schnittstelle aufweisen, wie sie im WSDL-Dokument festgelegt ist.

Wie bereits erwähnt, enthalten derartige WSDL-Dokumente keine Port- und Dienst-Definitionen. Diese Informationen werden in den entsprechenden UDDI-Datenstrukturen beschrieben. Zum Beispiel wird die tatsächliche Netzwerkadresse eines Dienstes im accessPoint-Element des dazugehörigen bindingTemplate beschrieben.

10.6 Dienstkomposition und -interaktion

In Abschnitt 10.2 haben wir schon kurz die Dienste Einheitenumrechnerdienst und TemperaturInFahrenheitDienst vorgestellt. TemperaturInFahrenheitDienst liefert, genauso wie der Temperaturdienst, die Temperatur zu einem gegebenen Zeitpunkt zurück. Allerdings, und das ist der einzige wirkliche Unterschied, liefert TemperaturInFahrenheitDienst sein Ergebnis im Gegensatz zum Temperaturdienst immer in Grad Fahrenheit. Es ist also sinnvoll, den Dienst TemperaturInFahrenheitDienst nicht von Grund auf neu zu implementieren, sondern auf die vorhandenen Dienste Temperaturdienst und Einheitenumrechnerdienst zurückzugreifen. Abbildung 10-7 zeigt, wie der TemperaturInFahrenheitDienst arbeitet. Der zusammengesetzte Dienst ruft, sobald er eine Anfrage erhält, den Temperaturdienst auf, um die Temperatur in Grad Celsius zu dem angegebenen Zeitpunkt zu ermitteln. Das Ergebnis des Temperaturdienstes muss er dann in geeigneter Form an den Einheitenumrechnerdienst schicken, damit dieser die Grad Celsius in Grad Fahrenheit umrechnet. Abschließend muss der TemperaturInFahrenheit-Dienst die Antwort auf die Anfrage im passenden Format zurückschicken.

Der Dienst TemperaturInFahrenheitDienst hat damit im Grunde nur zwei Aufgaben zu erfüllen. Zum einen muss er die beiden Basisdienste ansprechen und ihnen dabei die richtigen Parameter zur Verfügung stellen. Zum anderen muss er das Ergebnis des Temperaturdienstes auswerten, daraus die notwendigen Parameter für den Einheitenumrechnerdienst extrahieren und das Ergebnis des Einheitenumrechnerdienstes auswerten, um daraus sein eigenes Ergebnis zu ermitteln.

Um derartige Kompositionen von Web Services automatisieren zu können, wurden Sprachen basierend auf XML entwickelt. Diese erlauben die Spezifikation, welche Dienste wann angesprochen werden sollen und wie die Ergebnisdokumente eines Dienstes transformiert werden müssen, um als Eingabe für den

nächsten Dienst geeignet zu sein. Beispiele für solche Sprachen sind die Web Services Flow Language (WSFL) [Leym01] von IBM, XL [FlKo01] und XLang [That01] von Microsoft. Diese Sprachen unterstützen nicht nur die sequenzielle Verkettung von Diensten wie im obigen (einfachen) Beispiel, sondern erlauben die Spezifikation eines komplexen Ablaufs, ähnlich wie dies etwa bei Workflows möglich ist. Welche Möglichkeiten im Einzelnen bestehen, um Dienste zu kombinieren, hängt von der verwendeten Sprache ab.

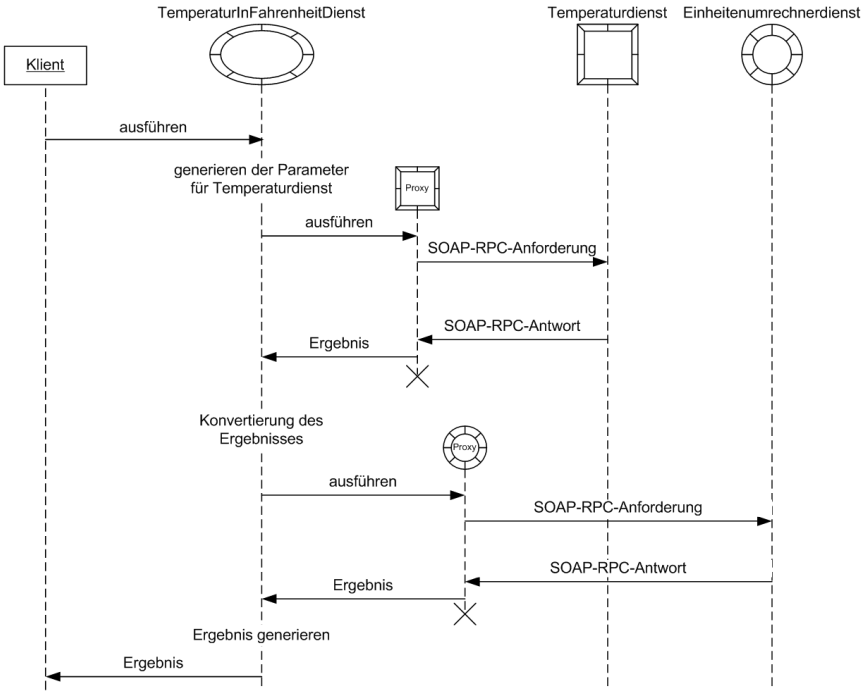


Abb. 10-7: *Dienstkomposition am Beispiel TemperaturInFahrenheitDienst*

Der TemperaturInFahrenheitDienst kann natürlich öffentlich zugänglich gemacht werden, analog zu den Diensten Temperaturdienst und Einheitenumrechnerdienst. Dazu ist es zum einen notwendig, ein passendes WSDL-Dokument zu erstellen. Dieses unterscheidet sich vom WSDL-Dokument des Temperaturdienstes, da eine Rückgabe der Einheit der Temperatur unnötig ist, weil die Einheit auf Grad Fahrenheit festgelegt ist. Zum anderen müssen die entsprechenden Einträge in UDDI eingefügt werden. Diese bestehen aus einem entsprechenden tModel, das auf das neue WSDL-Dokument für den Dienst TemperaturInFahrenheitDienst verweist, sowie geeigneten bindingTemplate- und businessService-Einträgen, falls diese noch nicht existieren. Alternativ hätte man auch das tModel des Temperaturdienstes verwenden können, allerdings müsste man in diesem Fall im Ergebnis des TemperaturInFahrenheitDienstes die Einheit angeben.

10.7 Plattformen, Produkte und Infrastrukturen

Aufgrund des Potenzials von Web Services, das Internet in eine Dienstplattform zu verwandeln, wird ihnen von Industrie und Forschung immer größere Aufmerksamkeit geschenkt. Produkte und Plattformen wie HP Web Services Platform, Sun ONE und Microsoft .NET zeigen dies. Die Produkte und Plattformen basieren auf den in den vorhergehenden Abschnitten vorgestellten Standards und Technologien wie SOAP, UDDI und WSDL. Sie stellen Werkzeuge zur Verfügung, mit denen z. B. existierende Anwendungen unkompliziert und schnell als Dienste zur Verfügung gestellt werden können, einschließlich der Generierung von entsprechenden WSDL-Dokumenten, passenden UDDI-Einträgen usw. Außerdem unterstützen sie die Komposition und Interaktion dieser Dienste und ermöglichen damit die Interaktion der dahinter stehenden Anwendungen. Auf diese Art und Weise ist es möglich, Business-to-Business-Integration zu realisieren. Dabei entwickelt man für vorhandene Anwendungen eine Web-Service-Schnittstelle, über die dann beispielsweise die ERP-Systeme verschiedener Firmen interagieren können.

Neben diesen kommerziellen Produkten und Plattformen existieren Forschungsprototypen wie etwa ServiceGlobe [KSSK02] und SELF-SERV [BDSN02]. Das ServiceGlobe-System soll im Folgenden als eine Beispielplattform beschrieben werden.

Das ServiceGlobe-System stellt eine neuartige Infrastruktur für die Implementierung und Ausführung von Web Services zur Verfügung. Es ermöglicht die Verwaltung von und die Suche nach Web Services, basierend auf den Standards UDDI und WSDL. Web Services können im Internet auf beliebigen Rechnern, die in die ServiceGlobe-Föderation integriert sind, verteilt ausgeführt und dynamisch aufgerufen werden. ServiceGlobe stellt unter anderem Standardfunktionalität von Dienstplattformen zur Verfügung, z. B. Kommunikation auf SOAP/XML-Basis sowie ein Transaktions- und Sicherheitssystem. Zusätzlich werden auch verschiedene Optimierungen wie etwa Lastverteilung unterstützt.

ServiceGlobe unterscheidet grundsätzlich zwischen zwei Arten von Diensten: externe und interne Dienste. Externe Dienste sind existierende, stationäre Dienste, wie sie zurzeit im Internet zu finden sind und die nicht von ServiceGlobe selbst angeboten werden. Derartige Dienste können auf beliebigen Systemen im Internet implementiert sein und beliebige Schnittstellen zum Aufruf anbieten. ServiceGlobe ermöglicht die Verwendung solcher Dienste unabhängig von ihrer tatsächlichen Schnittstelle, etwa SOAP oder RPC, mit Hilfe von Adaptoren. Diese kapseln die Kommunikation mit externen Diensten und bieten damit eine einheitliche Schnittstelle an. Damit erfüllen sie ähnliche Aufgaben wie Wrapper in Mediatorensystemen.

Interne Dienste sind in ServiceGlobe ausgeführte Dienste. Sie sind in Java implementiert, basierend auf dem API, das von ServiceGlobe zur Verfügung gestellt wird. ServiceGlobe verwendet XML zur Kommunikation mit anderen Diensten, d. h., ein Dienst empfängt ein einzelnes XML-Dokument als Eingabe und generiert ein einzelnes XML-Dokument als Ausgabe. Es existieren zwei Arten von in-

ternen Diensten: dynamische und statische Dienste. Statische Dienste sind ortsabhängig, d. h., sie können nicht dynamisch auf beliebigen ServiceGlobe-Servern ausgeführt werden. Derartige Dienste benötigen möglicherweise Zugriff auf bestimmte lokale Ressourcen, z. B. ein lokales DBMS, um den internen Zustand abzuspeichern, oder sie benötigen bestimmte Rechte, z. B. für den Zugriff auf das Dateisystem, die nur auf bestimmten Servern verfügbar sind. Diese Einschränkungen verhindern eine Ausführung von statischen Diensten auf beliebigen ServiceGlobe-Servern. Ein Beispiel für einen statischen Dienst ist der Temperaturdienst, der Zugriff auf einen Temperatursensor benötigt und damit nicht auf beliebigen Servern ausgeführt werden kann. Im Gegensatz dazu sind dynamische Dienste ortsunabhängig. Sie sind zustandslos, d. h., der interne Zustand derartiger Dienste wird nach der Abarbeitung einer Anfrage verworfen, und sie benötigen keine speziellen Ressourcen oder Rechte. Deshalb können sie auf jedem beliebigen ServiceGlobe-Server ausgeführt werden. Beispiele für dynamische Dienste sind der Einheitenumrechnerdienst und der TemperaturInFahrenheitDienst.

Es gibt eine orthogonale Klassifikation von internen Diensten in Adaptoren, einfache Dienste und zusammengesetzte Dienste. Adaptoren wurden bereits eingeführt. Einfache Dienste sind interne Dienste, die keinen anderen Dienst verwenden. Der Einheitenumrechnerdienst ist zum Beispiel ein einfacher Dienst. Zusammengesetzte Dienste verwenden andere Dienste, die in diesem Zusammenhang auch als Basisdienste bezeichnet werden. Ein Beispiel für einen zusammengesetzten Dienst ist der TemperaturInFahrenheitDienst. Natürlich kann ein zusammengesetzter Dienst auch selbst wieder von einem weiteren zusammengesetzten Dienst als Basisdienst verwendet werden.

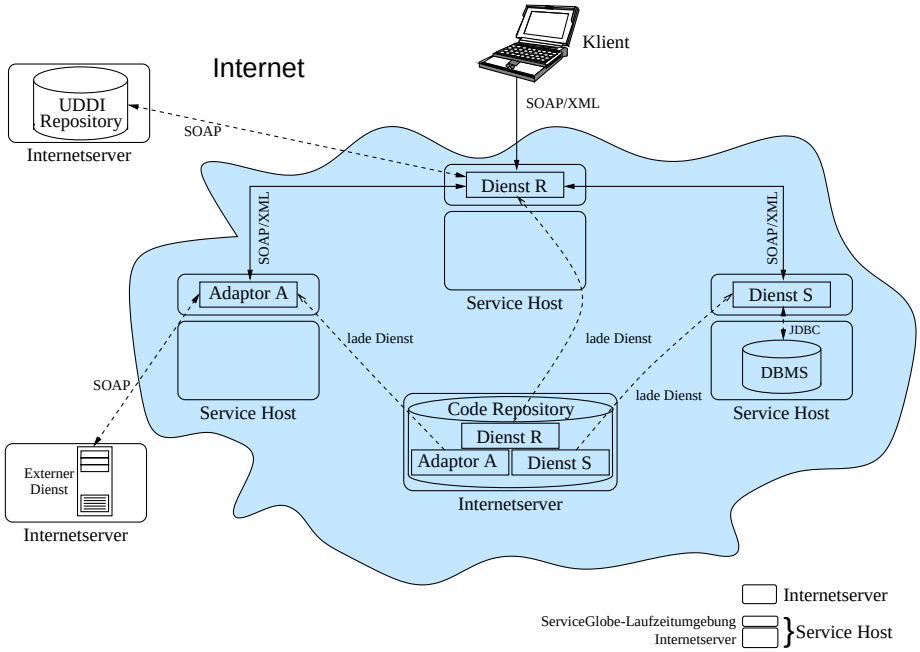


Abb. 10-8: *Architektur des ServiceGlobe-Systems*

Interne Dienste werden auf so genannten Service Hosts ausgeführt. Dabei handelt es sich um gewöhnliche Internet-Server, auf denen zusätzlich eine ServiceGlobe-Laufzeitumgebung läuft. Sie kommunizieren mit anderen Diensten mittels SOAP-Nachrichten. Die Kommunikation mit externen Diensten wird durch Adaptoren gekapselt, welche Anfragen auf die Schnittstelle des externen Dienstes umsetzen. Interne Dienste sind mobiler Code. Wenn ihr ausführbarer Code benötigt wird, wird er auf Anforderung von so genannten Code Repositories auf Service Hosts geladen. UDDI wird verwendet, um ein passendes Code Repository für einen bestimmten Dienst zu finden.

Abbildung 10-8 gibt einen Überblick über die wesentlichen Komponenten des ServiceGlobe-Systems. Zu Beginn sendet der Client (mit Hilfe von SOAP) einen Aufruf an einen Service Host, den dynamischen Dienst R auszuführen. Wenn der ausführbare Code dieses Dienstes noch nicht lokal gepuffert ist, wird er von einem Code Repository geladen und auf dem Service Host instanziiert. Während seiner Ausführung greift Dienst R auf die Basisdienste Adaptor A und Dienst S zu. Dazu werden die beiden Dienste zuerst mit Hilfe von UDDI gesucht. Wenn nötig, wird Adaptor A auf einen passenden Service Host geladen und dort im Auftrag von Dienst R ausgeführt. Dienst S ist ein statischer Dienst. Er läuft bereits und kann mit Hilfe von SOAP-Nachrichten angesprochen werden. Im Folgenden wird nun die Ausführung eines internen Dienstes in ServiceGlobe beschrieben, die sich grob in vier Schritte aufteilen lässt:

Dienstspezifikation: Unter Dienstspezifikation versteht man den Prozess der Programmierung eines Dienstes. Eine Möglichkeit dazu ist die Verwendung von Java und dem API von ServiceGlobe. Komfortabler ist es allerdings, eine spezialisierte Programmiersprache, z. B. XL [FlKo01], oder ein Werkzeug zur grafischen Dienstkomposition (ähnlich wie bei Workflows) zu verwenden. Damit man in ServiceGlobe ausführbare Dienste erhält, muss sowohl ein Programm als auch eine grafische Repräsentation in eine oder mehrere Java-Klassen kompiliert werden, welche das ServiceGlobe-API verwenden.

Dynamische Dienstauswahl: UDDI ordnet jedem Dienst ein tModel zu, das im Grunde die Semantik und die Schnittstellen des Dienstes beschreibt. Ein Dienst ist somit eine Implementierung seines tModels. In ServiceGlobe müssen zusammengesetzte Dienste nicht unbedingt einen konkreten Dienst aufrufen. Es reicht, wenn sie das tModel angeben oder »aufrufen«. Eine passende Implementierung wird dann während der Kompilierung oder zur Laufzeit ausgewählt. Dieser Vorgang der Auswahl eines Dienstes für ein tModel wird dynamische Dienstauswahl genannt. Dynamische Dienstauswahl ist nicht darauf beschränkt, nur eine Implementierung für ein tModel auszuwählen. Da UDDI bei einer Anfrage alle Dienste zurückliefert, die ein gegebenes tModel implementieren, können mehrere Implementierungen ausgewählt und folglich auch mehr als ein Dienst aufgerufen werden. Dadurch ist es zum Beispiel möglich, auf die schnellste Antwort zu warten.

Dienstverteilung: Der Hauptvorteil von dynamischen Diensten ist ihre Ortsunabhängigkeit. Zur Laufzeit ermöglicht dies die dynamische Verteilung solcher Dienste auf beliebige Service Hosts, wodurch sich eine Vielzahl an Optimierungsmöglichkeiten ergibt. Mehrere Instanzen eines dynamischen Dienstes können auf verschiedenen Rechnern ausgeführt werden, um Lastverteilung und Parallelisierung zu erreichen. Dynamische Dienste können auf den Service Hosts ausgeführt werden, die optimale Voraussetzungen aufweisen, z. B. einen schnellen Prozessor, ausreichend Hauptspeicher oder eine schnelle Netzwerkverbindung. Zusammen mit dem Laden von Diensten zur Laufzeit ergibt sich eine große Flexibilität im Hinblick auf Lastbalancierung und Optimierung. ServiceGlobe besitzt ein umfassendes Sicherheitssystem, das die Sicherheitsaspekte von mobilem Code berücksichtigt und somit Service Hosts vor bösartigen Diensten schützt. Ausführliche Beschreibungen von Sicherheitssystemen, die dem von ServiceGlobe gewählten Ansatz ähnlich sind, finden sich in [BKKK01, SeBK01, BrKK99] und [KrIK00, KSKK99].

Dienstladen zur Laufzeit: Nach ihrer Verteilung müssen dynamische Dienste von Code Repositories geladen und auf den ausgewählten Service Hosts ausgeführt werden. Dadurch ist die Menge der verfügbaren Dienste nicht fest, sondern sie kann zur Laufzeit durch jeden Teilnehmer der ServiceGlobe-Föderation erweitert werden.

10.8 Zusammenfassung und Ausblick

In diesem Kapitel haben wir die wichtigsten Standards und Technologien im Bereich Web Services vorgestellt. Web Services verwandeln das Internet immer mehr zu einer Plattform, auf der Dienste angeboten werden, und sie ermöglichen vollautomatische Dienstnutzung und Dienstkomposition. Anhand eines einfachen Beispiel-Web-Dienstes, einem Temperaturdienst für die Stadt Passau, haben wir die wichtigsten Standards im Bereich Web Services für Datenaustausch, Dienstverwaltung und Dienstbeschreibung erklärt: SOAP, UDDI, WS-Inspection und WSDL.

SOAP ist ein Kommunikationsprotokoll für verteilte Anwendungen, das es ermöglicht, strukturierte und typisierte Daten mit Hilfe von XML auszutauschen. UDDI ist ein globales Verzeichnis für Dienst-Metadaten, das die Speicherung von Dienst-Metadaten in einer einheitlichen Datenstruktur (UDDI-Schema) an zentralen und öffentlich zugänglichen Stellen im Internet (UDDI-Server) durch einheitliche Veröffentlichungs-Mechanismen ermöglicht. Außerdem unterstützt UDDI die Metadatenabfrage durch eine normierte Anfragesprache. Der WS-Inspection-Standard definiert, wie Dienstanbieter Metadaten ihrer Dienste auch – alternativ oder zusätzlich zu der Speicherung in einem UDDI-Verzeichnis – in standardisierter Form auf ihrem Web-Server anbieten können, indem sie dort entsprechende WS-Inspection-Dokumente ablegen. WSDL schließlich legt fest, auf welche Weise man die zum Aufruf eines Dienstes notwendigen Informationen in einem XML-

Dokument spezifiziert. Zu diesen Informationen zählen z. B. die Operationen, die der Dienst unterstützt, das Format der Eingabe- und Ausgabedaten und das Protokoll, das zur Kommunikation mit dem Dienst verwendet werden muss.

Neben diesen Basistechnologien sind wir auch darauf eingegangen, wie Web Services die Komposition von neuen, zusammengesetzten Diensten unter Verwendung von bestehenden Diensten erleichtern. Wir haben anhand des Beispieldienstes TemperaturInFahrenheitDienst gezeigt, wie einfach es ist, einen neuen Dienst aus bestehenden Diensten zusammenzubauen. Neuartige, spezialisierte Sprachen zur Dienstkomposition, wie z. B. WSFL, XL und XLang, werden dies in Zukunft noch einfacher machen. Abschließend haben wir als eine Beispielplattform das ServiceGlobe-System beschrieben.

Noch ist die Entwicklung im Bereich Web Services im Anfangsstadium. Das kann man unter anderem daran erkennen, dass selbst die grundlegenden Standards SOAP, UDDI und WSDL noch Änderungen unterworfen sind. Sprachen zur Dienstkomposition sind gerade erst im Entstehen begriffen. Gleiches gilt für Standards, die das Schema der Ein- und Ausgabedaten für bestimmte Dienste, z. B. im Bereich B2B, festlegen. Ein Beispiel hierfür ist die Universal Business Language (UBL).[11] Das zuständige Komitee hat sich zum Ziel gesetzt, eine Standardbibliothek für XML-Dokumente aus dem Geschäftsbereich zu definieren, z. B. für Bestellungen und Rechnungen. Doch trotz dieser Probleme zeigt gerade das Engagement vieler bedeutender IT-Firmen, dass Web Services sich durchsetzen werden.

Literatur

[ABDK02] Atkinson, B.; Brown, A.; Della-Libera, G.; Kaler, C.; Klein, J.; Konersmann, S.; Leach, P.; Manferdelli, J.; Shewchuk, J.; Simon, D.: *Web Services Security Language* (WS-Security). http://msdn.microsoft.com, 2002.

[BBMN01] Ballinger, K.; Brittenham, P.; Malhotra, A.; Nagy, W. A.; Pharies, S.: *Web Services Inspection Language (WS-Inspection) 1.0*. http://www.ibm.com/developerworks/ webservices/library/ws-wsilspec.html, 2001.

[BDSN02] Benatallah, B.; Dumas, M.; Sheng, Q. Z.; Ngu, A. H. H.: *Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services*. In: Proceedings of the 18th International Conference on Data Engineering IEEE Computer Society (ICDE), 2002, S. 297-308.

[BEKL00] Box, D.; Ehnebuske, D.; Kakivaya, G.; Layman, A.; Mendelsohn, N.; Nielsen, H.F.; Thatte, S.; Winer, D.: *Simple Object Access Protocol (SOAP) 1.1*. http://www.w3c.org/TR/SOAP/, 2000.

[BKKK01] Braumandl, R.; Keidl, M.; Kemper, A.; Kossmann, D.; Kreutz, A.; Seltzsam, S.; Stocker, K.: *ObjectGlobe: Ubiquitous Query Processing on the Internet*. In: The VLDB Journal: Special Issue on E-Services, Volume 10, Nummer 3, 2001, S. 48-71.

[BrKK99] Braumandl, R.; Kemper, A.; Kossmann, D.: *Database Patchwork on the Internet: Project Demo*. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, 1999, S. 550-552.

11. Siehe `http://www.oasis-open.org/committees/ubl/`

[CCMW01] Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S.: *Web Services Description Language (WSDL) 1.1*. http://www.w3.org/TR/wsdl/, 2001.

[CuER01] Curbera, F.; Ehnebuske, D.; Rogers, D.: *Using WSDL in a UDDI Registry.* http://www.uddi.org/bestpractices.html, 2001.

[Dada96] Dadam, P.: *Verteilte Datenbanken und Client/Server-Systeme*. Springer-Verlag, Berlin, 1996.

[FlKo01] Florescu, D.; Kossmann, D.: *An XML Programming Language for Web Service Specification and Composition*. In: IEEE Data Engineering Bulletin, Volume 24, Nummer 2, 2001, S. 48-56.

[KeEi01] Kemper, A.; Eickler, A.: *Datenbanksysteme. Eine Einführung*. Oldenbourg-Verlag, 4. erweiterte Auflage, 2001.

[KeWi01] Kemper, A.; Wiesner, C.: *HyperQueries: Dynamic Distributed Query Processing on the Internet*. In: Proceedings of the 27th International Conference on Very Large Data Bases (VLDB), 2001, S. 551-560.

[KKKK01] Keidl, M.; Kreutz, A.; Kemper, A.; Kossmann, D.: *Verteilte Metadatenverwaltung für die Anfragebearbeitung auf Internet-Datenquellen*. In: Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), 9. GI-Fachtagung, 2001, S. 107-126.

[KKKK02] Keidl, M.; Kreutz, A.; Kemper, A.; Kossmann, D.: *A Publish & Subscribe Architecture for Distributed Metadata Management*. In: Proceedings of the 18th International Conference on Data Engineering IEEE Computer Society (ICDE), 2002, S. 309-320.

[KrIK00] Krivokapic, N.; Islinger, M.; Kemper, A.: *Migrating Autonomous Objects in a WAN Environment*. In: Journal of Intelligent Information Systems, Volume 15, Nummer 3, 2000, S. 221-252.

[KSKK99] Keidl, M.; Seltzsam, S.; Kemper, A.; Krivokapic, N.: *Sicherheit in einem Java-basierten verteilten System autonomer Objekte*. In: Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), 8. GI-Fachtagung, 1999, S. 38-58.

[KSSK02] Keidl, M.; Seltzsam, S.; Stocker, K.; Kemper, A.: *ServiceGlobe: Distributing E-Services across the Internet*. Projektvorführung auf der 28th International Conference on Very Large Data Bases (VLDB), 2002.

[KSWD02] Kraiß, A.; Schön, F.; Weikum, G.; Deppisch, U.: *Mit HEART zu Middleware-basierten Antwortzeitgarantien für E-Services*. In: Datenbank-Spektrum, Band 20, Heft 1, 2002, S. 64-74.

[Leym01] Leymann, F.: *Web Services Flow Language (WSFL 1.0)*. http://www.ibm.com/ software/solutions/webservices/pdf/WSFL.pdf, 2001.

[ScSt00] Scribner K.; Stiver M.C.: *Understanding SOAP.* Sams Publishing, 2000.

[SeBK01] Seltzsam, S.; Börzsönyi, S.; Kemper, A.: *Security for Distributed E-Service Composition*. In: Proceedings of the 2nd International Workshop on Technologies for E-Services (TES), 2001, S. 147-162.

[That01] Thatte, S.: *XLANG - Web Services for Business Process Design*. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/, 2001.

[UDDI] *Universal Description, Discovery and Integration (UDDI)*. http://www.uddi.org.

[WiWK02] Wiesner, C.; Winklhofer, P.; Kemper, A.: *Building Dynamic Market Places Using HyperQueries*. Projektvorführung auf der VIII. Conference on Extending Database Technology (EDBT), 2002.

11 Data-Warehouse-Einsatz zur Web-Zugriffsanalyse

Erhard Rahm, Thomas Stöhr

Kurzfassung

Die Analyse des Nutzungsverhaltens von Websites ermöglicht wichtige Hinweise zur Optimierung und Weiterentwicklung eines Web-Auftritts. Skalierbarkeit und Flexibilität der Auswertungen verlangen oft eine datenbankbasierte Realisierung. Wir diskutieren hierzu verschiedene Varianten, insbesondere den Einsatz eines »Web Data Warehouse«, in dem neben den Web-Log-Daten Informationen zu Nutzern/Kunden, Inhalten/Produkten und Anwendungsfunktionen integriert werden. Weiterhin geben wir einen Überblick zu derzeit verfügbaren Werkzeugen für die Web-Zugriffsanalyse.

11.1 Einführung

Eine aussagekräftige quantitative Analyse und Bewertung der Nutzung von Websites wird immer wichtiger. Durch eine derartige *Web-Zugriffsanalyse* sollen die vom Web-Server protokollierten Zugriffe sowie weitere Informationen ausgewertet werden, um ein möglichst genaues Bild vom Zugriffsverhalten der Nutzer einer Website zu erhalten. Mit den Auswertungen wird die Antwort auf eine Vielzahl von technischen, inhaltlichen und nutzerbezogenen Fragestellungen angestrebt, u.a.

- Wie gut ist die Leistung der Website (Durchsatz, Antwortzeit, Datenvolumen)?
- Welche Seiten / Inhalte / Produkte interessieren die Nutzer (bzw. Kunden) am meisten, welche werden dagegen kaum genutzt bzw. nicht gefunden?
- Wie ist die zeitliche Entwicklung beim Zugriff (aktuell im Vergleich zur Vorwoche, Vormonat etc.)?
- Woher kommen die Besucher (Suchmaschine, Portal, Anklicken eines Werbe-Banners etc.)?
- Wie ist das Navigationsverhalten innerhalb der Website (wichtige Pfade innerhalb von Sitzungen, Einstiegs- und Ausstiegsseiten)?
- Wer kommt auf die Website?
- Wie hoch ist die Rückkehrquote von Nutzern?
- Wie hoch ist die Kaufquote?
- Wie zufrieden sind die Nutzer? Warum haben sie die Site verlassen?

- Welche Informationen bzw. Produkte werden häufig zusammen betrachtet bzw. gekauft?
- Wie wirken sich Änderungen in der Strukturierung und im Design einer Website aus?
- Wie wirken sich Marketing-Aktivitäten (z.B. Banner-Werbung, Print-Anzeige etc.) im Zugriffsverhalten aus?
- Wie ist die Wirtschaftlichkeit der Website (Return on Investment)?

Die Ergebnisse der Web-Zugriffsanalyse sollen helfen, die Zielsetzungen eines Web-Auftritts besser zu erreichen. Damit bestimmen diese Zielsetzungen die benötigten Auswertungen oder Bewertungsmetriken und auch den Realisierungsaufwand einer Web-Zugriffsanalyse. Ein umfassender Bewertungsansatz ist besonders kritisch für E-Commerce-Unternehmen (Online-Buchhändler, Online-Versandhandel, Online-Kaufhäuser, ...). Sie benötigen möglichst genaues Wissen über die Interessen und Bedürfnisse ihrer Nutzer und Kunden, um diese optimal bedienen zu können (Gewinnung neuer Kunden, Ausbau der Kundenbindung). Aber auch andere Unternehmen und nicht-kommerzielle Einrichtungen können eine Web-Zugriffsanalyse nutzen, um generelle Ziele wie einfache und schnelle Bereitstellung relevanter Informationen, Gewinnung neuer Nutzer, Erhöhung der Rückkehrquote von Nutzern etc. besser zu erreichen.

In den letzten Jahren hat das Thema der Web-Zugriffsanalyse in Forschung und Praxis große Bedeutung erlangt [SCDT00, Spil00, WKDD01]. Eine Vielzahl von Werkzeugen für bestimmte Aspekte ist bereits verfügbar, insbesondere zur Auswertung der von den Web-Servern geführten *Web-Log-Dateien,* in denen nahezu jeder Maus-Klick der Website-Nutzer registriert wird. Neben der von uns verwendeten Bezeichnung der Web-Zugriffsanalyse findet man viele weitere Begriffe, u.a. *Web Usage Mining*, *Clickstream-Analyse*, *Web Traffic-Analyse*, *E-Analytics*, *E-Intelligence* etc. Diese Namen orientieren sich dabei teilweise an den ausgewerteten Daten (z.B. Clickstream-Log), den Analyseverfahren (z.B. einfache »Traffic«-Bewertungen, »Business Intelligence«-Auswertungen oder Data-Mining-Analysen) und dem Anwendungsschwerpunkt (z.B. E-Commerce).

In diesem Kapitel geben wir einen Überblick zu wesentlichen Realisierungsaspekten einer umfassenden Web-Zugriffsanalyse und den derzeit verfügbaren Werkzeugen. Dazu stellen wir im nächsten Kapitel einfache dateibasierte Ansätze einer Datenbank-Lösung gegenüber und plädieren für den Einsatz eines Data-Warehouse-Ansatzes, der durch Integration unterschiedlicher Datenquellen auch inhaltliche und benutzerbezogene Auswertungen ermöglicht. Zur Umsetzung des Warehouse-Ansatzes diskutieren wir zunächst die relevanten Datenquellen sowie wichtige Teilaufgaben bei der Datentransformation (Bereinigung von Log-Daten, Bestimmung von Sitzungen, Nutzerzuordnung). Anschließend stellen wir ein einfaches Datenbankschema (Stern-Schema) vor, das vielfältige Auswertungen zulässt, und geben einen Überblick über wesentliche Auswertungsansätze, insbesondere Data-Mining-Verfahren. Nach einer Diskussion von Möglichkeiten zur Umsetzung der Analyseergebnisse folgt der Überblick zu derzeitigen Werkzeugen.

11.2 Datei- vs. datenbankbasierte Realisierungsansätze

Einfache Werkzeuge und Verfahren zur Web-Zugriffsanalyse werten lediglich die Web-Log-Dateien des Web-Servers aus. Damit erstellt man im Wesentlichen vordefinierte statistische Auswertungen auf den Daten, die in diesen Log-Dateien pro Zugriff vermerkt werden (s.u.). Beispiele sind zeitliche Entwicklung der Zugriffshäufigkeit und -verteilung auf bestimmte Seiten (URLs) und Rechner (IP-Adressen), transferierte Datenmengen, Häufigkeit fehlerhafter Zugriffe etc. Die dateibasierten Ansätze ermöglichen zwar bereits eine Reihe von Erkenntnissen, weisen jedoch wesentliche Beschränkungen auf:

- *Datenmengen:* Die Auswertungen auf Dateiebene erfordern das sequenzielle Lesen der kompletten Log-Daten und sind somit nur für kleinere Datenmengen ausreichend schnell erstellbar. Größere Websites produzieren jedoch sehr große Mengen an Daten – bis zu mehrere Gigabytes pro Tag. Für zeitliche Vergleichsanalysen sollten diese Daten über Zeiträume von einigen Jahren genutzt werden können, was leicht zu Datenmengen im höheren Terabyte-Bereich führt.
- *Flexibilität der Auswertungen:* Die vordefinierten Auswertungsberichte können nur Basisinformationen liefern, nicht jedoch die gezielte Analyse einzelner Aspekte. Erforderlich ist ein Spektrum an Analysemöglichkeiten von vordefinierten Berichten, interaktiven Abfragen bis hin zu Data-Mining-Analysen.
- *Inhaltliche und benutzerbezogene Auswertungen:* Die Web-Log-Daten unterstützen primär technische Auswertungsaspekte, jedoch keine ausreichenden inhaltlichen (z.B. produktbezogene) und nutzer/kundenspezifischen Auswertungen. Diese erfordern die Verknüpfung mit weiteren Daten wie Angaben zu Struktur und Inhalten einer Website sowie Nutzer-/Kundenmerkmalen.

Die Behebung dieser Nachteile verlangt eine leistungsfähige Datenbank-Lösung, da nur so eine hohe Skalierbarkeit und Flexibilität der Auswertungen erreicht werden kann. Je nach dem angestrebten Auswertungsumfang kommen hierbei wiederum mehrere Varianten in Frage, u.a.:

- Verwendung einer *»einfachen« Datenbank* zur Speicherung und Auswertung der Web-Log-Daten (ohne Integration weiterer Daten),
- Verwendung eines *dedizierten Data Warehouse* zur Web-Zugriffsanalyse, wobei neben den Web-Log-Daten weitere Datenquellen für inhaltliche und nutzerspezifische Auswertungen integriert werden,
- Einbettung der Web-Zugriffsanalyse in ein *Unternehmens-Data-Warehouse*.

Der erstgenannte Ansatz verursacht den geringsten Realisierungsaufwand und unterstützt bereits größere Datenmengen und bessere Auswertungsmöglichkeiten als dateibasierte Verfahren. Allerdings können die Auswertungsziele nur durch die Integration mehrerer Datenquellen umfassend erreicht werden, wie von Data-Warehouse-Lösungen ermöglicht. Der Data-Warehouse-Ansatz [BaGü01] hat sich in den letzten Jahren unabhängig von dem hier betrachteten Anwendungsge-

biet als Schlüsseltechnologie für entscheidungsunterstützende Analysen insbesondere in größeren Unternehmen etabliert. Kennzeichnende Merkmale sind, dass die auszuwertenden Daten aus mehreren Datenquellen extrahiert und im Rahmen einer separaten Datenbank integriert werden. Dieses Data Warehouse wird periodisch (z.B. täglich) aus den Quelldaten aktualisiert und ist von seiner Struktur und Realisierung auf Analyseerfordernisse ausgerichtet. Im Rahmen so genannter *OLAP*-Auswertungen (Online Analytical Processing) können interaktiv auch auf sehr großen Datenmengen eine Vielzahl von Auswertungen erfolgen. Daneben werden weitere Auswertungen wie Erstellung vordefinierter Berichte oder Data-Mining-Analysen unterstützt.

Durch Integration verschiedener Datenquellen sowie die umfassenden Auswertungsmöglichkeiten sind Data-Warehouse-Lösungen für die Web-Zugriffsanalyse besonders geeignet [KiMe00, StRQ00, ScNL01]. Eine Variante ist dabei die Einbettung der Web-Nutzungsdaten innerhalb eines Unternehmens-Data-Warehouse zur Realisierung eines umfassenden *Customer Relationship Management (CRM)*, bei dem die Beziehungen zwischen Kunden und Unternehmen über alle Interaktionskanäle (Web, E-Mail, Telefon, klassische Post, ...) hinweg erfasst, analysiert und genutzt werden [Schr00]. Die CRM-Spezifika sollen jedoch im Weiteren nicht vertieft werden.

Abb. 11-1 zeigt die Grobarchitektur eines datenbankbasierten Ansatzes zur Web-Zugriffsanalyse mit dediziertem Data Warehouse (»Web Data Warehouse«). Die protokollierten Zugriffe auf der Website werden kontinuierlich transformiert und in das Warehouse überführt. Daneben erfolgt die Integration von Angaben zu Kunden bzw. Nutzern, der Site-Struktur, zu Produkten/Inhalten etc. Auf dem integrierten und periodisch aktualisierten Datenbestand des Web Data Warehouse erfolgen dann unterschiedliche Auswertungen sowie Data-Mining-Analysen, z.B. zur Entdeckung von Mustern im Navigationsverhalten, zur Analyse des Kaufverhaltens und zur Segmentierung von Kunden. Die Umsetzung der Analyseerkenntnisse kann zu Anpassungen im Web-Auftritt oder zur Durchführung bestimmter Marketing-Aktivitäten führen, deren Auswirkungen dann wieder im Rahmen der Web-Zugriffsanalyse überprüfbar sind. Dieser geschlossene Kreislauf (»closed loop«) ist wesentlich für den Erfolg und verdeutlicht die Vorteile des Interaktionskanals »Web«, der eine schnelle Reaktion auf geänderte Verhältnisse und eine kontinuierliche Kontrolle und Analyse der Effektivität gestattet.

Die Vorteile einer Data-Warehouse-Lösung verlangen allerdings auch einen relativ hohen Realisierungsaufwand, insbesondere bei der Aufbereitung und Integration der Daten. Auf die dabei anfallenden Probleme wird im Folgenden näher eingegangen.

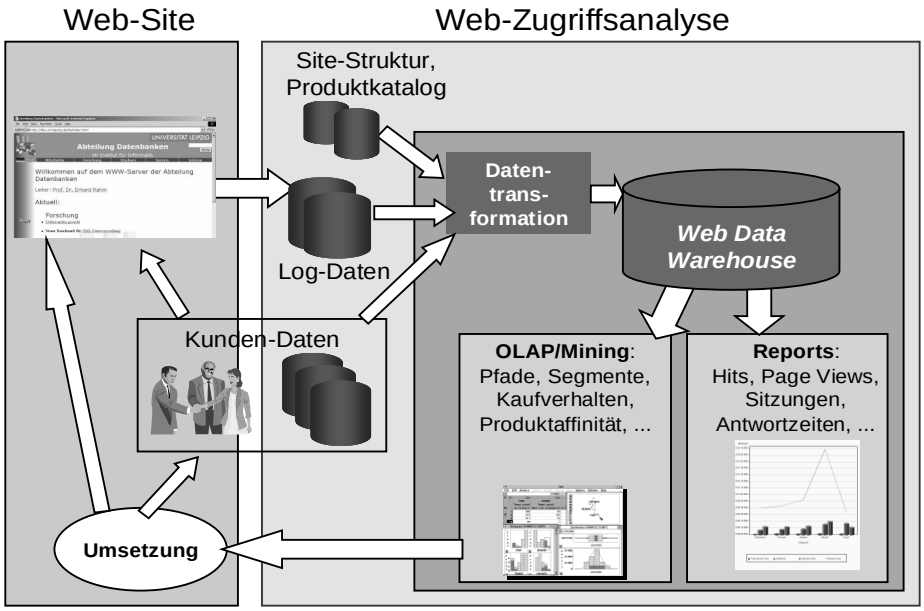


Abb. 11-1: *Grobarchitektur einer Data-Warehouse-basierten Web-Zugriffsanalyse*

11.3 Datenquellen und Datentransformation

Zur Umsetzung eines Warehouse-Ansatzes zur Web-Zugriffsanalyse betrachten wir in diesem Abschnitt die wesentlichen Datenquellen sowie die wichtigsten Schritte der Datentransformation. Hierzu diskutieren wir zunächst den Aufbau der Web-Log-Dateien sowie die Verwendung weiterer Datenquellen (Site-Struktur, Nutzerinformationen, Inhaltskategorisierung etc.). Zur Datentransformation werden Aufgaben der Datenbereinigung, Sitzungs- und Nutzeridentifikation behandelt.

11.3.1 Aufbau der Web-Log-Dateien

Die Zugriffe auf die Inhalte einer Website werden von Web-Servern verwaltet, welche jeden von ihnen bearbeiteten Dateizugriff oder »Hit« (Aufruf einer HTML-Seite, Zugriff auf Bilddatei, Skript-Ausführung, ...) in einer Log-Datei protokollieren. Jeder Zugriff wird durch einen Log-Satz in einem bestimmten Format repräsentiert, wobei vor allem das Common Log Format (CLF) sowie dessen Erweiterungen weit verbreitet sind. Ein Beispieleintrag dafür ist

```
green.dresdnerbank.de - - [11/Jul/2002:15:33:13 +0200]        // Host, Zeitpunkt
"GET /skripte/WMS/inhalt2.html HTTP/1.0" 200 7885             // Request, Status, #Bytes
"http://www.google.de/search?q=workflow&hl=de&meta=" "        // Referrer
Mozilla/4.0 (compatible; MSIE 5.01; Windows NT; drebaIE-ZE)"  // Nutzer-Agent
```

Wesentliche Komponenten der Einträge sind:

- *Host:* Name bzw- IP-Adresse des Rechners, von dem der Zugriff erfolgte,
- *Zeitpunkt* des Zugriffs (Datum, Uhrzeit, Zeitzone),
- *HTTP-Request*, der insbesondere die URL der aufgerufenen Datei enthält sowie möglicherweise Aufrufparameter im Fall von Skripten und dynamischen Webseiten,
- *Status*: standardisierte Kennzeichnung des Zugriffsstatus, z.B. 200 (ok), 304 (not modified since last retrieval) oder 404 (not found),
- *Bytes*: übertragene Datenmenge (bei Status 304 in der Regel 0, da eine aktuelle Version des Objekts in einem Cache vorliegt),
- *Referrer:* URL der Seite, von der aus der Nutzer zugegriffen hat; diese Seite kann auf derselben Website oder extern liegen sowie (für dynamische Seiten) Aufrufparameter enthalten, z.B. die Suchbegriffe bei der Ergebnisseite einer Suchmaschine (s. Beispieleintrag),
- *Nutzer-Agent:* Kennzeichnung des Browsers und des Betriebssystems des Nutzers.

Die beiden letzten Angaben sind nicht im CLF-Standard enthalten, werden jedoch von den meisten Web-Servern unterstützt. Daneben können diese noch für eine Protokollierung weiterer Angaben wie Cookies (s.u.) oder anwendungsspezifische Erweiterungen konfiguriert werden.

11.3.2 Weitere Datenquellen

Zur strukturellen, nutzerbezogenen und inhaltlichen Auswertung der Web-Zugriffe ist eine Verknüpfung mit Informationen weiterer Datenquellen erforderlich, u.a:

- Angaben zur *Struktur* der Website, insbesondere zur Vernetzung der Webseiten (Hypertext-Struktur) sowie von Navigationsmöglichkeiten aufgrund dynamischer Anfragen.
- *Nutzerbezogene Informationen*, wie bereits angelegte Nutzerprofile oder Kundenangaben aus Unternehmensdatenbanken. Möglicherweise können auch allgemeine demografische Informationen herangezogen werden, z.B. um aus Angaben wie Adressen Rückschlüsse auf Einkommen etc. zu gewinnen.
- Angaben zur *Anwendungsfunktion* einzelner Webseiten bzw. Seitenbereiche. Dies erfordert zum einen eine auf die Auswertungserfordernisse ausgelegte Kategorisierung von Anwendungsfunktionen und zum anderen eine Zuordnung der Webseiten bzw. Seitenbereiche zu diesen Kategorien. Die Kategorisierung kann eine einfache Aufzählung sein oder eine mehrstufige (hierarchische) Aufgliederung. So könnte z.B. eine Kategorisierung von Webseiten nach Navigations- und Inhaltsseiten erfolgen und bei den Inhaltsseiten eine Unterteilung nach Funktionen wie Produktansicht, Bestellseite etc.

- Angaben zu den *Inhalten* einzelner Webseiten bzw. dynamischer Anfragen. Analog zur Festlegung der Seitenfunktion ist hier eine Kategorisierung der Inhalte und die Zuordnung zu Webseiten (Web-Zugriffen) erforderlich. Bei einem Online-Shop empfiehlt sich z.B. eine mehrstufige Kategorisierung der Produkte analog zu einem Produktkatalog, um Auswertungen bezüglich unterschiedlicher Produktgruppen zu unterstützen (z.B. Anteil der Zugriffe auf Fachbücher vs. Belletristik, Bestellungshäufigkeit für Computer-Bücher gegenüber allen Fachbüchern etc.). Die inhaltliche Zuordnung von Web-Zugriffen zu den Inhaltskategorien erfordert oft die Berücksichtigung dynamischer Zugriffsparameter (z.B. Produktnummer).
- Werden die aufgerufenen Funktionen weitgehend unter Kontrolle von Applikations-Servern durchgeführt, sind zur funktionalen und inhaltlichen Auswertung der Website-Nutzung die *Log-Dateien der Applikations-Server* als Datenquelle zusätzlich auszuwerten.

Als Alternative zu den Web-Log-Dateien kommt die Aufzeichnung der Datentransfers auf Netzwerkebene durch so genannte *Packet Sniffer* als Datenquelle in Frage. Dieser Ansatz kann auch bei größeren Websites mit mehreren Web-Servern sämtliche Zugriffe in der chronologischen Reihenfolge erfassen und somit ein aufwändiges Mischen mehrerer Log-Dateien umgehen. Andererseits gibt es Beschränkungen, z.B. können verschlüsselte Nachrichtenpakete nicht interpretiert werden und Web-Server-Techniken wie die Auswertung von Cookies werden erschwert.

Die von den Web-Servern registrierbaren Seitenzugriffe repräsentieren das Zugriffsverhalten nicht vollständig. Denn zur Reduzierung des zu übertragenden Datenvolumens sowie zur Unterstützung schneller Antwortzeiten erfolgt an mehreren Stellen auf dem Weg zwischen dem Browser eines Benutzers zum Web-Server eine Pufferung (Caching) von Webseiten, insbesondere durch die Browser selbst sowie bei vermittelnden Rechnerknoten, z.B. Proxy-Rechnern in Unternehmen oder beim Internet Service Provider (ISP). Entsprechende Aufzeichnungen der Zugriffe bei den Browsern bzw. Proxy-Rechnern stellen somit prinzipiell weitere Datenquellen zur Web-Zugriffsanalyse dar. Ein Problem dabei ist die potenziell große Anzahl solcher Datenquellen, die zudem weitgehend außerhalb der Kontrolle eines Website-Betreibers liegen. Insbesondere dürften nur wenige Benutzer bereit sein, ihre Web-Zugriffe z.B. durch ein entsprechendes Browser-Plugin aufzeichnen zu lassen und diese Zugriffsdaten für externe Auswertungen zur Verfügung zu stellen. Eine Abmilderung des Problems besteht darin, Heuristiken zur Erkennung von Zugriffslücken im Log zu verwenden und eine so genannte *Pfadvervollständigung* durchzuführen (s.u.).

11.3.3 Datentransformationen

Bei der Übernahme und Integration der Daten aus den verschiedenen Quellen in das Web Data Warehouse sind umfassende Datentransformationen erforderlich.

Dies betrifft insbesondere die Web-Log-Einträge, auf deren Behandlung wir uns in diesem Abschnitt konzentrieren wollen. Die Angaben aus den weiteren Datenquellen können vergleichsweise einfach in das Schema des Web Data Warehouse (Kapitel 11.4) integriert werden. Zur Datentransformation von Web-Log-Einträgen besprechen wir im Weiteren die Datenbereinigung, die Bestimmung von Sitzungen, die Nutzeridentifikation sowie die Pfadvervollständigung. Diese Aufgaben sollten durch ein Werkzeug behandelt werden, das die gängigen Web-Log-Formate unterstützt. Bei mehreren Web-Servern sind die Log-Dateien zu mischen. Dies erfolgt am einfachsten über die Zeitstempel der Log-Einträge, vorausgesetzt die Uhrzeiten der Web-Server sind eng synchronisiert [KiMe00].

Die Notwendigkeit der Datentransformationen erkennt man bereits daran, dass die Anzahl der Log-Einträge und damit der Hits nur eine sehr grobe Angabe zur Nutzungsintensität einer Website und ihrer Bereiche entsprechen, obwohl diese Metrik noch häufig Verwendung findet. Dies hängt u.a. damit zusammen, dass diese Zahl sehr stark von Zugriffen z.B. durch Roboter-Programme verfälscht sein kann. Ausserdem besteht eine große Abhängigkeit zum jeweiligen Web-Design, da jedes in eine Seite eingebettete Bild (z.B. Button), Skript etc. einen eigenen Hit verursacht. Diese Effekte müssen im Rahmen der Datentransformation bereinigt werden, um auch Kennzahlen wie die Anzahl der vollständig angezeigten Seiten (Page Views), die Anzahl der Sitzungen (Besuche) sowie die Anzahl der Nutzer (Besucher) bestimmen zu können. Abb. 11-2 verdeutlicht, dass diese Größen in 1:n-Beziehungen zueinander stehen, d.h., pro Besucher gibt es eine oder mehrere Sitzungen, pro Sitzung einen oder mehrere Page Views und pro Page View ein oder mehrere Hits. Weiterhin ist die Anzahl der Sitzungen und Nutzer deutlich aussagekräftiger für die Nutzungsintensität als die Anzahl der Hits.

Abb. 11-2: *Von vielen Hits zu wenigen Besuchern*

Zur Verdeutlichung stellt Abb. 11-3 diese und einige weitere aus den Web-Log-Daten abgeleitete technische Größen für zwei einfache Websites gegenüber. Man erkennt, dass damit schon einige Beurteilungen der Websites unterstützt werden. So weist der erste Server (Lehrstuhl) pro Tag fast die fünffache Anzahl von Hits, jedoch weniger Besuche als der zweite Server (Dokumenten-Server) auf, da in ersterem Fall (aufgrund des Designs) durchschnittlich mehr als doppelt so viele Hits pro Seite und fast dreimal so viele Seiten pro Sitzung anfallen. Die beim zweiten Server primär abgerufenen, relativ voluminösen Dokumente führen dort zu einem höheren Gesamtdatenvolumen. Die Anzahl der übertragenen Dateien ist generell kleiner als die Anzahl der Hits, wegen der Nutzung von Caches (Status 304) und Fehlern beim Zugriff (Datei nicht vorhanden etc.). Für den ersten Server war der

Caching-Effekt besonders ausgeprägt (begünstigt durch die große Zahl von Hits pro Sitzung), wo nur für 70% der Hits eine Dateiübertragung stattfand.

	dbs.uni-leipzig.de (Lehrstuhl Datenbanken)	dol.uni-leipzig.de (Dokumenten-Server)
#Hits pro Tag	10000	2100
#übertragene Dateien pro Tag	7000	1900
#Seiten (page views) pro Tag	3200	1400
#Sitzungen pro Tag	430	540
Datenvolumen in MB pro Tag	77	190
#Hits / Seite	3,1	1,5
#Seiten / Sitzung	7,4	2,6
KB / Datei	11	100

Abb. 11-3: *Bewertungskennzahlen zweier Beispiel-Websites (Jan. 2002)*

Datenbereinigung (Data Cleaning)

Hierbei sind alle für die Auswertungen irrelevanten bzw. verfälschenden Zugriffe zu identifizieren und ggf. zu eliminieren, wodurch der Datenumfang oft signifikant reduziert werden kann. Dies betrifft typischerweise die Zugriffe auf in den explizit angeforderten Seiten eingebetteten Hilfsdateien (*auxiliary resources*) wie Bilder und andere Multimedia-Inhalte, Stylesheet-Dateien, Applets, Skripte etc. Weiterhin sind die Zugriffe von Roboter-Programmen (Crawlern), die z.B. zur Unterstützung von Suchmaschinen (Kap. 7) das Web ständig durchkämmen, zu identifizieren. Hierzu gibt es mehrere Möglichkeiten, z.B. das Vorliegen bestimmter IP-Adressen und Browser-Bezeichnungen oder das Erkennen einer für menschliche Nutzer unerreichbar hohen Zugriffsfrequenz. Auch von Site-spezifischen Indexierungsprogrammen verursachte Log-Einträge sowie Zugriffe lokaler Benutzer sind möglicherweise zu extrahieren. Die jeweils notwendigen Filteraktionen sind site- und auswertungsabhängig und sollten somit durch das Transformationswerkzeug konfigurierbar sein. Weitere Bereinigungsaktionen betreffen die Vereinheitlichung und Ergänzung von Datensätzen wie zur Namensauflösung für numerische IP-Adressen (*reverse DNS lookup*), um z.B. 139.18.2.117 durch db1.informatik.uni-leipzig.de zu ersetzen.

Sitzungs- und Nutzeridentifikation

Um das Nutzungsverhalten von Websites bewerten zu können, ist die Bestimmung von Sitzungen unabdingbar. Eine Sitzung besteht dabei aus der Sequenz von Web-Zugriffen eines Nutzers. Somit erfordert die Bestimmung der Sitzungen bereits eine Zuordnung von Log-Einträgen zu Nutzern. Dann rechnet man üblicherweise alle Zugriffe eines Nutzers, deren zeitlicher Abstand einen Grenzwert (z.B. 30 Minuten) nicht übersteigt, derselben Sitzung zu. Mit dieser Vorgehensweise wird die Aufgabe der Sitzungsidentifikation im Wesentlichen auf das Problem der Nutzeridentifikation abgebildet. Diese ist jedoch im Web-Kontext ein

großes Problem, da die überwiegende Zahl der Zugriffe anonym erfolgt und das verwendete HTTP-Protokoll zustandslos arbeitet, d.h. jeder Web-Zugriff eines Nutzers wird unabhängig von seinen vorhergehenden abgewickelt.

Die Nutzeridentifikation ist offenbar über die Sitzungsidentifikation hinaus von großer Bedeutung für die Web-Zugriffskontrolle, da nur die Berücksichtigung möglichst aller Sitzungen eines Nutzers ein aussagekräftiges Bild über seine Interessen und Nutzungsgewohnheiten gestattet. Dies erfordert insbesondere, dass ein Nutzer bei einer neuen Sitzung wiedererkannt werden kann. Die weitreichendsten Erkenntnisse werden möglich, wenn der Nutzer darüber hinaus auch personifiziert werden kann, d.h. seine personenbezogenen Daten wie Name oder E-Mail-Adresse bekannt sind (z.B. ein wiederkehrender Kunde). Wir unterscheiden somit drei Stufen der Nutzeridentifikation mit zunehmender Aussagekraft, aber auch wachsender Schwierigkeit der Umsetzung:

- temporäre Nutzeridentifikation für die Dauer einer Sitzung,
- sitzungsübergreifende Nutzeridentifikation, insbesondere Erkennung wiederkehrender Nutzer,
- Personifizierung.

In den ersten beiden Fällen bleibt der Nutzer namentlich unbekannt und somit anonym.

Ansatz	sitzungsbezogene Nutzeridentifikation	sitzungsübergreifende Nutzeridentifikation	Personifizierung	Abdeckungsgrad
IP-Adresse + Agent + Referrer	o / +	–	– –	++
temporäre Cookies	+	n.a.	n.a.	+
Session-IDs in URL / versteckten Feldern	+	n.a.	n.a.	+
persistente Cookies	+	+	o	+
Nutzer-Registrierung	++	++	++	–

Abb. 11-4: *Grobbewertung von Ansätzen zur Nutzeridentifikation*
(++ sehr gut, + gut, o mittel, -- schlecht, -- sehr schlecht, n.a. nicht anwendbar)

Im Folgenden diskutieren wir Realisierungsansätze für die drei Arten der Nutzeridentifikation. Abb. 11-4 zeigt hierzu eine zusammenfassende Bewertung hinsichtlich der Eignung für die einzelnen Identifizierungsvarianten (Genauigkeit der Identifizierung). Als weitere Bewertungsgröße wird zur Nutzbarkeit der Ansätze noch der erreichbare Abdeckungsgrad abgeschätzt, d.h., welcher Anteil der Zugriffe ist aufgrund von notwendiger Nutzerkooperation bzw. erforderlichen Browser-Einstellungen mit einem Ansatz überhaupt behandelbar. Man erkennt bereits, dass eine korrekte Identifizierung aller Nutzer unmöglich ist und dass die

Verfahren (und damit die Werkzeuge, die sie einsetzen) sich in Genauigkeit und Abdeckungsgrad teilweise stark unterscheiden. Die drei erstgenannten Ansätze eignen sich nur zur sitzungsbezogenen Nutzeridentifikation.

Temporäre Nutzeridentifikation / Sitzungsidentifikation

Einige einfache Auswertungsansätze verwenden lediglich die Host-Angabe (IP-Adresse) zur Nutzeridentifikation, d.h., alle Log-Sätze mit derselben Host-Adresse werden einem Nutzer zugerechnet. Dies ist jedoch selbst für die temporäre Nutzeridentifikation sehr ungenau, da – gerade zur Anonymisierung – IP-Adressen zunehmend für jeden Zugriff dynamisch generiert werden (z.B. von Internet-Service-Providern). Eine wesentlich bessere Zuordenbarkeit wird erreicht, wenn neben der IP-Adresse (ggf. reduziert auf den Teil, der nicht dynamisch variiert wird) noch die Nutzer-Agent-Angabe (Browser und Betriebssystem) sowie die Referrer-Information hinzugenommen werden. Damit würden im Beispiel von Abb. 11-5 die (in eine relationale Repräsentation überführten) Einträge 27101, 27103 und 27104 demselben Nutzer und derselben Sitzung zugeordnet, trotz der dynamischen Variation im Präfix der Host-Angabe.

Nr	Host	Zeitpunkt	URL	Referrer	Browser	Betriebssystem
27101	cw06.a1.srv.t-online.de	... 16:27:13	A	google.de	MSIE 5.5	NT 5.0
27102	proxy2.sbs.de	... 16:27:14	A	-	MSIE 6.0	Windows98
27103	cw08.a1.srv.t-online.de	... 16:27:43	B	A	MSIE 5.5	NT 5.0
27104	cw10.a1.srv.t-online.de	... 16:28:02	C	B	MSIE 5.5	NT 5.0

Abb. 11-5: *Sitzungsidentifikation mit dynamischen IP-Adressen (Beispiel)*

Alternativen bezüglich der temporären Nutzeridentifikation bzw. Sitzungsidentifikation sind die Generierung einer expliziten Sitzungs-ID durch den Web-Server und deren Austausch mit dem Browser bei jeder folgenden Interaktion. Hierzu sind folgende Ansätze gebräuchlich:

- Ablage der ID innerhalb eines temporären Cookies (Session Cookie), das vom Browser auf dem Rechner des Nutzers gespeichert und vom Web-Server bei jedem erneuten Zugriff gelesen wird. Die Lebensdauer dieser Cookies ist eng begrenzt und läuft mit Beendigung einer Browser-Nutzung ab.
- Speicherung der ID innerhalb der an den Browser zurückgelieferten HTML-Seiten, entweder in versteckten Eingabefeldern oder durch dynamische Modifikation aller in der Seite vorkommenden URLs.

Die ID-Werte werden vom Web-Server beim erneuten Zugriff des Nutzers identifiziert und im Web-Log protokolliert und können somit zur temporären Nutzeridentifikation herangezogen werden. Wesentlicher Nachteil dieser Ansätze ist der vom Web-Server zu leistende Zusatzaufwand, der auch die Zugriffszeiten für den

Benutzer verlängert. Ausserdem gibt es Abhängigkeiten zu Browser-Einstellungen (z.B. Akzeptanz von temporären Cookies).

Sitzungsübergreifende Nutzeridentifikation

Der am weitesten verbreitete Ansatz hierbei ist der Einsatz *persistenter Cookies*. Die Cookies enthalten insbesondere eine eindeutige Nutzer-ID und werden dauerhaft auf der Festplatte des Nutzer-Rechners gespeichert. Über das vom Web-Server gelesene Cookie kann somit ein wiederkehrender Nutzer erkannt und z.B. eine zugeschnittene Nutzung der Website ermöglicht werden (keine wiederholte Eingabe von Präferenzen, Kennwörtern etc.). Dies erfordert seitens der Website die Verwaltung von über Cookies identifizierten Nutzerprofilen, die jedoch die Anonymität der Nutzer wahren, solange keine personenbezogenen Angaben erfasst werden. Eine Einschränkung bei Cookies ist, dass sie nur einen Browser auf einem bestimmten Rechner, also nicht unbedingt eine bestimmte Person kennzeichnen (Nutzung des Rechners durch mehrere Personen). Ausserdem kann von Personen, die mehrere Rechner bzw. Browser nutzen, nur ein Teil der Zugriffe korrekt zugeordnet werden. Schließlich wird der Grad der erreichbaren Nutzeridentifikation dadurch begrenzt, dass die Cookies seitens der Nutzer abgelehnt bzw. gelöscht werden können. Derzeit akzeptieren die weitaus meisten Web-Nutzer offenbar das Anlegen persistenter Cookies, jedoch mehren sich die Bedenken aus Datenschutzgründen. Die Website-Betreiber sollten daher zur Vertrauensbildung im eigenen Interesse die Inhalte und Verwendung der angelegten Nutzerprofile offenlegen, auch wenn keine personenbezogenen Angaben gespeichert werden.

Eine zuverlässigere Alternative zur sitzungsübergreifenden Nutzeridentifikation ist die *Nutzer-Registrierung*, welche anonym (unter einem Pseudonym) oder personenbezogen erfolgen kann. Dabei muss sich der Nutzer registrieren und bei jeder neuen Sitzung durch Angabe des Registrierungskennworts explizit identifizieren (während einer Sitzung kann der Web-Server die Kennung wie im Falle der Sitzungs-IDs mit dem Browser austauschen). Die explizite Identifizierung ist für die Nutzer sehr lästig und wird somit meist nur von einem kleinen Anteil (potenzieller) Nutzer akzeptiert. Auch ist bei Verwendung von Pseudonymen eine Mehrfach-Registrierung möglich.

Personifizierung

Die personenbezogene Identifizierung wird meist über eine Nutzer-Registrierung mit Angabe persönlicher Informationen wie Name, Anschrift, E-Mail-Adresse etc. erreicht. Sie ist notwendig für Geschäftsbeziehungen wie Online-Bestellungen, Online-Banking etc., bei der die Nutzer also einen Kundenstatus haben. Der Vorteil liegt in der eindeutigen Identifizierung und der weitestgehenden Nutzbarkeit personenbezogener Angaben (Alter, Geschlecht, Beruf, Wohnort, Kaufverhalten etc.) bei der Web-Zugriffsanalyse. Auf der anderen Seite sind viele Nutzer

zur Preisgabe der Anonymität nur bereit, wenn dies zur Abwicklung von Online-Diensten unabdingbar ist bzw. deutliche Vorteile daraus resultieren. Somit wird auf Grundlage einer Nutzerregistrierung nur ein geringer Abdeckungsgrad bezogen auf alle Web-Zugriffe erreicht werden können.

Auch persistente Cookies können zur Personifizierung genutzt werden, wenn der Nutzer-ID personenbezogene Daten zugeordnet werden, z.B. nach einer erfolgten Bestellung oder einer personenbezogenen Registrierung. Damit kann auch ohne explizite Anmeldung ein wiederkehrender (registrierter) Kunde erkannt und eine personenbezogene Erfassung des Nutzungsverhaltens erreicht werden. Solche Verwendungsformen personenbezogener Daten erfordern in Deutschland aus Datenschutzgründen die explizite Zustimmung des Nutzers.

Pfadvervollständigung

Wie erwähnt spart die Pufferung von Webseiten durch Browser oder Proxy-Rechner Zugriffe beim Web-Server ein, so dass es zu »Lücken« in den Log-Aufzeichnungen kommt. Um eine Verfälschung der Analyseergebnisse zu begrenzen, kann versucht werden, solche Lücken bei der Log-Transformation zu erkennen und im Rahmen der so genannten Pfadvervollständigung zu korrigieren [CoMS99]. Die Erkennung der Lücken basiert einerseits auf der Strukturinformation einer Website und zum anderen auf der durch die Referrer-Information dokumentierten Sequenz der protokollierten Zugriffe. So muss im Beispiel von Abb. 11-6 der im Log erkannte Pfad A-B-C-D eine Lücke zwischen C und D enthalten, da gemäß der Site-Struktur keine direkte Verbindung zwischen diesen Seiten besteht.

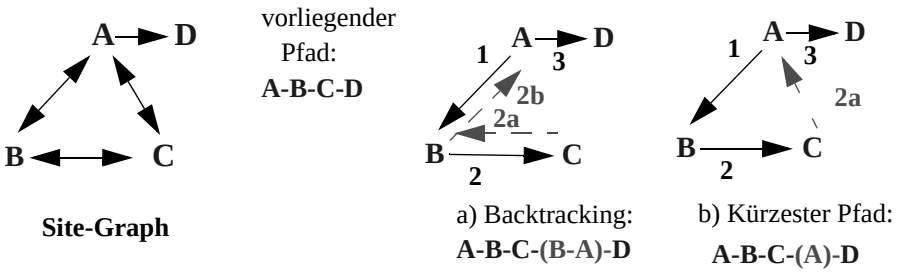


Abb. 11-6: *Pfadvervollständigung (Beispiel)*

Die Schließung erkannter Lücken kann nur in Annäherung durch Einsatz von Heuristiken verfolgt werden. Ein Ansatz ist ein so genanntes *Backtracking*, bei dem unterstellt wird, dass die Lücken primär durch die Rückwärtsnavigation über die Back-Funktion des Browsers enstanden sind. Dies ist insbesondere dann wahrscheinlich, wenn der Log-Eintrag zu einer unverbundenen Zielseite (D im Beispiel) in der Referrer-Information eine Seite angibt, die bereits in der jüngsten Vergangenheit referenziert wurde (z.B. A). Zur Korrektur werden in diesem Fall Zugriffe zur Rückwärtsnavigation auf die entsprechende Referrer-Seite eingefügt, im Beispiel zwei Zugriffe auf B und A (Reaktion a in Abb. 11-6). Alternativ kann

unter allen Pfaden, die gemäß Site-Struktur möglich sind, eine Auswahl unter bestimmten Optimalitätskriterien erfolgen, z.B. kürzester oder häufigster Pfad. Im Beispiel könnte so mit nur einem zusätzlichen Zugriff die Lücke geschlossen werden (Reaktion b in Abb. 11-6).

11.4 Data-Warehouse-Schema

Kennzeichnend für auf relationalen Datenbanken basierende Data Warehouses ist eine spezielle Strukturierung der Daten zur Unterstützung mehrdimensionaler Auswertungen. Große Bedeutung hat hierbei der Einsatz so genannter Stern-Schemata gefunden, deren Einsatz sich auch zur Web-Zugriffsanalyse eignet [KiMe00]. Dabei ist die Masse der auszuwertenden Daten im Rahmen von Faktentabellen gespeichert. Die zur Auswertung benötigten beschreibenden Eigenschaften der Fakten sind durch Dimensionstabellen repräsentiert; die Verbindung zwischen den Tabellen erfolgt durch Fremdschlüssel der Faktentabelle, die sich auf jeweils eine Dimension beziehen.

Zur Illustration zeigt Abb. 11-7 die Grobstruktur eines möglichen Stern-Schemas zur Web-Zugriffsanalyse mit einer Faktentabelle und acht Dimensionstabellen, welche die meisten der besprochenen Daten integrieren. Als Granularität der Faktentabelle und damit der Auswertungen werden einzelne Seitenzugriffe (Page Views) verwendet, d.h., für jeden nach der Bereinigung verbliebenen Zugriff auf eine Webseite gibt es einen Satz in der Faktentabelle. Für die Auswertung relevante Merkmale aus den Log-Sätzen wie Tag und Zeit der Zugriffe, Referrer, Seite (URL) und Nutzerangaben (IP-Adresse, Browser, Betriebssystem, Cookie-Inhalte) sind direkt durch entsprechende Dimensionstabellen repräsentiert. Bei den Seiten kann auch eine Kategorisierung nach Funktion (Navigation vs. Inhalt, Produktinformation, Bestellung ...) abgebildet werden. Zur inhaltlichen Kategorisierung wird, wie v.a. für E-Commerce-Anwendungen bedeutsam, die Produkttabelle verwendet, welche einen Produktkatalog repräsentiert. Somit kann für jeden Web-Zugriff, z.B. abgeleitet aus dynamischen Parametern wie der Produkt-ID, eine gezielte Zuordnung zu einzelnen Produkten erfolgen. Die Dimension Session-Typ erlaubt eine Kategorisierung von Sitzungen, z.B. nach Dauer, Länge oder zur Unterscheidung verschiedener Arten von Informations- oder Kaufbesuchen. Die Dimensionstabellen sind i.A. hierarchisch organisiert, so dass Auswertungen auf unterschiedlichen Granularitätsstufen möglich sind (z.B. zeitliche Auswertungen auf Tages-, Monats-, Quartals- oder Jahresebene; Referrer-Analyse auf Seiten- oder Site-Ebene, Produktauswertungen für Produktgruppen oder einzelne Produkte etc.).

Die Nutzertabelle soll nur Angaben zu anonymen Besuchern erfassen und kann auch aus den Sitzungen ableitbare Nutzerprofile enthalten (Häufigkeit und Intensität der Nutzung, Interessensschwerpunkte etc.). Personifizierte Nutzer wie Kunden eines Unternehmens sollen in einer eigenen Kundentabelle verwaltet werden, um besondere Auswertungen für diese wichtige Personengruppe zu unter-

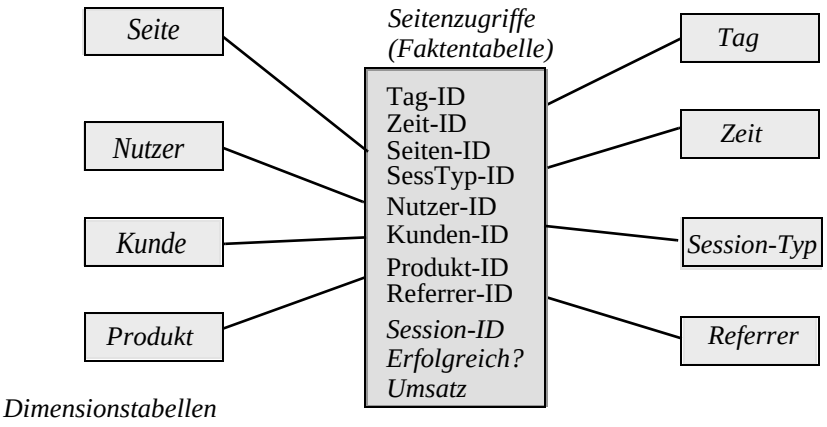


Abb. 11-7: *Stern-Schema zur Web-Zugriffsanalyse (Beispiel)*

stützen. Für die Auswertung interessieren dabei u.a. die Zuordnung der Kunden zu Kategorien bezüglich Wohnort/Region, Geschlecht, Altersgruppe, Familienstand, Berufsgruppe, Umsatz etc.

Die Faktentabelle enthält neben den Verweisen (Fremdschlüssel) auf die Dimensionseinträge noch so genannte Kennzahlen (kursiv dargestellt). In dem Beispiel werden sie genutzt, um die Zugehörigkeit von Seitenzugriffen zu einer bestimmten Sitzung zu dokumentieren und um anzuzeigen, ob der Zugriff durch den Web-Server erfolgreich durchgeführt wurde. Für E-Commerce-Anwendungen kann ferner der z.B. durch eine Bestellung entstehende Umsatz abgelegt werden, dessen Analyse natürlich von besonderer Bedeutung ist.

Es versteht sich von selbst, dass je nach Zielsetzung einer Website und ihrer Analyse weitere bzw. andere Merkmale Berücksichtigung finden können. Soll z.B. die Effektivität bestimmter Werbemaßnahmen oder Design-Änderungen analysiert werden, müssten solche Maßnahmen auch innerhalb von entsprechenden Dimensionstabellen repräsentiert werden [KiMe00]. Auch können zusätzliche Faktentabellen verwendet werden, etwa für eine genauere Sitzungsanalyse oder zur Vorberechnung häufig benötigter Auswertungen (z.B. monatliche oder kundenbezogene Zusammenfassungen).

11.5 Analyse

Mit den im Web Data Warehouse integrierten Daten wird eine Vielzahl von Analysen unterstützt, insbesondere zur Beantwortung von technischen, inhaltlichen und nutzerbezogenen Fragestellungen, wie sie bereits in der Einleitung angesprochen wurden. Als Auswertungsverfahren können die umfangreichen von Data Warehouses unterstützten Ansätze genutzt werden, wie die Erstellung vordefinierter Berichte und Statistiken, die interaktive Durchführung von Anfragen und mehrdimensionaler OLAP-Auswertungen sowie komplexere Data-Mining-Ana-

lysen. Insbesondere bietet die Warehouse-Lösung eine hohe Auswertungsflexibilität, da unterschiedliche Auswahlkriterien auf verschiedenen Detaillierungsstufen nahezu beliebig kombinierbar sind. So kann die Referrer-Analyse, Nutzeranalyse oder inhaltliche Analyse unter Berücksichtigung der anderen Dimensionen erfolgen. Beispiele für solch kombinierte Anfragen sind: Von welchen Referrern kommen die Interessenten bestimmter Produkte, wie ist die zeitliche Entwicklung bei der Nutzungsintensität von Kunden vs. anonymen Nutzern, welche Produkte werden von jungen weiblichen Kunden besonders nachgefragt etc.

Zur Illustration zeigt Abb. 11-8 die grafische OLAP-Ausgabe zur inhaltlichen Auswertung von Zugriffen auf die Website eines Lehrstuhls. Die Ausgabe bezieht sich auf die in [StRQ00] vorgestellte Warehouse-Realisierung, bei der eine mehrstufige inhaltliche Kategorisierung der Webseiten zunächst nach Lehre, Forschung etc. erfolgt, bei der Lehre erfolgt eine weitere Unterteilung nach Vorlesungen, Übungen, Praktika etc. Die Beispielausgabe gibt die Zugriffshäufigkeit auf die Materialien zu einzelnen Vorlesungen verschiedenener Quartale an. Wie für OLAP-Auswertungen üblich können zur Verfeinerung/Vergröberung interaktiv Drill-down- bzw. Roll-up-Auswertungen entlang der Inhaltsdimension erfolgen oder neue Parameter für andere Auswertungsdimensionen (Zeitraum, Nutzer) gewählt werden.

Generell sind als Kennzahlen bzw. Bewertungsmetriken statistische Aggregierungen zu absoluten und relativen Zugriffshäufigkeiten, Verweilzeiten und Übertragungsvolumen hinsichtlich der unterschiedlichen Dimensionen relevant. Zur Fehlererkennung interessieren die Seiten und Wege, die zu einem nicht erfolgreichen Zugriff oder einem Sitzungsabbruch geführt haben. Wichtig sind ferner unterschiedliche *Konversionsraten*, z.B. welche Anteile der Nutzer bestimmte Inhalte aufgesucht bzw. bestimmte Aktivitäten durchgeführt haben. So interessieren

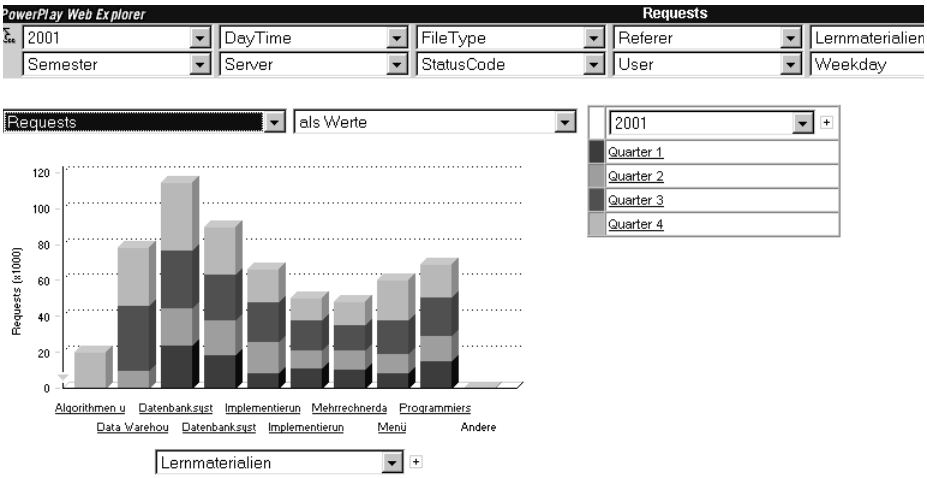


Abb. 11-8: *OLAP-Ausgabe zur inhaltlichen Zugriffsanalyse (Zugriffshäufigkeit auf Vorlesungsunterlagen einer Lehrstuhl-Website)*

im E-Commerce-Umfeld vor allem die Konversionsraten von Besuchern zu Kunden (Käufern), die Rückkehrquote von Kunden (auch ein Indikator für die Zufriedenheit) sowie der pro Kunde generierte Umsatz und Gewinn. Die Detailauswertung dieser Metriken kann dann genutzt werden, um profitable (bzw. unprofitable) Referrer, Produkte, Kunden etc. zu bestimmen. Zur Erklärung und Optimierung der Konversionsraten ist es wichtig, Navigationspfade im Rahmen von Besuchen näher zu bewerten, insbesondere die Übergangswahrscheinlichkeiten (Mikrokonversionsraten) für Zwischenstationen auf dem Weg von einer Einstiegsseite bis zur Bestellung oder anderen Ausstiegsseiten. Die Referrer-Analyse kann helfen, die Effektivität von Marketing-Aktivitäten wie Werbung auf externen Websites zu bewerten. Idealerweise kann sogar der Return on Investment bestimmt werden, in dem die Kosten einer Werbemaßnahme ins Verhältnis zu den neu generierten Umsätzen und Gewinnen gesetzt wird.

Data-Mining-Auswertungen sollen zusätzliche Erkenntnisse ermöglichen, indem nicht nur festgelegte Auswertungen berechnet, sondern relevante Muster im Nutzungsverhalten selbstständig entdeckt werden. Von besonderem Interesse sind hierzu vor allem Ansätze zur Segmentierung von Kunden und Nutzern hinsichtlich Kriterien wie z.B. Interessensgebieten und Kaufverhalten. Die Identifikation von Inhalten/Web-Bereichen mit ähnlichen Nutzungsmerkmalen kann zur Optimierung des Web-Auftritts genutzt werden. Für solche Mining-Aufgaben kommen unterschiedliche Techniken der Statistik und der künstlichen Intelligenz in Frage, insbesondere Cluster-Verfahren, Klassifikationsansätze sowie Assoziations- und Sequenzregeln [EsSa00]. So erlauben Cluster-Verfahren die Bestimmung von Nutzergruppen mit ähnlichen Interessen oder von ähnlichen Sitzungen, sofern geeignete Ähnlichkeitsmaße hierfür gefunden werden. Klassifikationsansätze ermöglichen z.B. die Vorhersage unbekannter Kundenmerkmale bzw. des künftigen Kundenverhaltens, indem das Wissen zu bekannten Kunden verallgemeinert wird. Assoziationsregeln unterstützen eine Warenkorbanalyse bzw. allgemein die Bestimmung von häufig zusammen aufgerufenen Inhalten (zusammen gekauften Produkten). Sequenzregeln analysieren, welche Inhalte oder Aktionen oft in aufeinander folgenden Sitzungen eines Benutzers aufgerufen werden.

11.6 Nutzung

Die aus den Analysen gewonnenen Erkenntnisse können vor allem zur Umstrukturierung einer Website verwendet werden, um erkannte Defizite zu beheben bzw. erkannte Optimierungspotenziale umzusetzen. Bereits die Auswertung anonymer Web-Zugriffe, also ohne Personifizierung, ermöglicht viele Verbesserungen, etwa Beseitigung fehlerhafter Verweise, verbesserte Nutzerführung durch angepasste Such- und Navigationsmöglichkeiten, Ergänzung und Aktualisierung der Inhalte und Dienste etc. Eine wichtige Rolle zur Nutzerführung nehmen gezielte Empfehlungen (Links auf andere Webseiten) und Produktangebote sowie deren Platzierung ein. Während eine statische Platzierung für alle Nutzer einfach realisierbar

ist, wird durch eine dynamische Generierung in Abhängigkeit vorhergehender Zugriffe oder erkannter Nutzermerkmale (z.B. für wiederkehrende Nutzer) eine weit größere Flexibilität erzielt. Bei Personifizierung der Nutzer und erstellten Nutzungsprofilen kann dies am weitestgehenden durch personenbezogene (individuelle) Empfehlungen und Angebote umgesetzt werden, wie z.B. im Rahmen von One-to-One-Marketing angestrebt. Generell sollen im E-Commerce-Umfeld durch die Empfehlungen/Angebote vor allem Anreize für erhöhten Umsatz und Gewinn (Cross-Selling und Up-Selling) gesetzt werden.

Diese Maßnahmen beziehen sich auf die Nutzer, welche die Website bereits erreicht haben. Darüber hinaus liegt eine wichtige Nutzungsart der Web-Zugriffsanalyse in verbesserten Marketing-Maßnahmen, um überhaupt mehr Nutzer auf die Website zu bringen und um neue Kunden zu gewinnen. Hierzu kann insbesondere die Referrer-Analyse wichtige Hinweise geben, auf welchen externen Websites Werbehinweise die besten Erfolgsaussichten versprechen. Ferner lassen sich aus personalisierten Nutzungsprofilen nach bestimmten Kriterien Adressaten für Marketing-Kampagnen (z.B. E-Mail- oder Briefwerbung) bestimmen.

Die Verwendung personenbezogener Daten zur Personalisierung einer Website oder für Marketing-Zwecke sollten nur bei expliziter Zustimmung der betroffenen Personen erfolgen. In Deutschland ist diese Forderung auch in den Datenschutzgesetzen festgeschrieben, wenngleich dies schwer zu überwachen ist. Allerdings sollte es auch im Interesse der Unternehmen liegen, ihren Kunden zu zeigen, dass sie keinen Datenmissbrauch zu befürchten haben. Hierzu gehört die Bekanntgabe von detaillierten Datenschutzrichtlinien bezüglich der Web-Nutzung (Cookie-Einsatz, was wird ausgewertet und wie genutzt etc.), das Einholen der Nutzereinwilligung bezüglich der Verwendung personenbezogener Daten für die Website-Personalisierung oder für Werbemaßnahmen sowie das jederzeitige Widerrufsrecht bezüglich der Zustimmung zu diesen Verwendungsformen.

11.7 Werkzeug-Markt

Der Markt für Werkzeuge zur Web-Zugriffsanalyse ist umfangreich und unübersichtlich. Eine Vielzahl von Anbietern aus den Bereichen Daten(bank)-Management, Business Intelligence oder Data Mining haben ihre Produktpalette um spezifische Web-Zugriffsanalyse-Werkzeuge, -Lösungen oder -Funktionalitäten ergänzt. Daneben existieren zahlreiche Spezialanbieter sowie Prototypen aus Forschungsgruppen, die z.B. spezielle Data-Mining-Algorithmen zur Pfadanalyse realisieren. Eine Auswahl der Anbieter zeigt Abb. 11-9.

Die Bandbreite der Funktionalitäten von Web-Zugriffsanalyse-Werkzeugen ist ebenfalls weit gefächert. Die einfachsten Werkzeuge liefern lediglich tabellarische Statistiken, die sie aus der Log-Datei eines Web-Servers erstellen, z.B. die Anzahl der Zugriffe/Besucher über die Zeit, die am häufigsten referenzierten Seiten/Pfade etc. Dadurch sind einfache Aussagen über die Anzahl durchgeführter »Clicks«, den Zeitverlauf der Web-Server-Last, oder »beliebte« Seiten zu treffen.

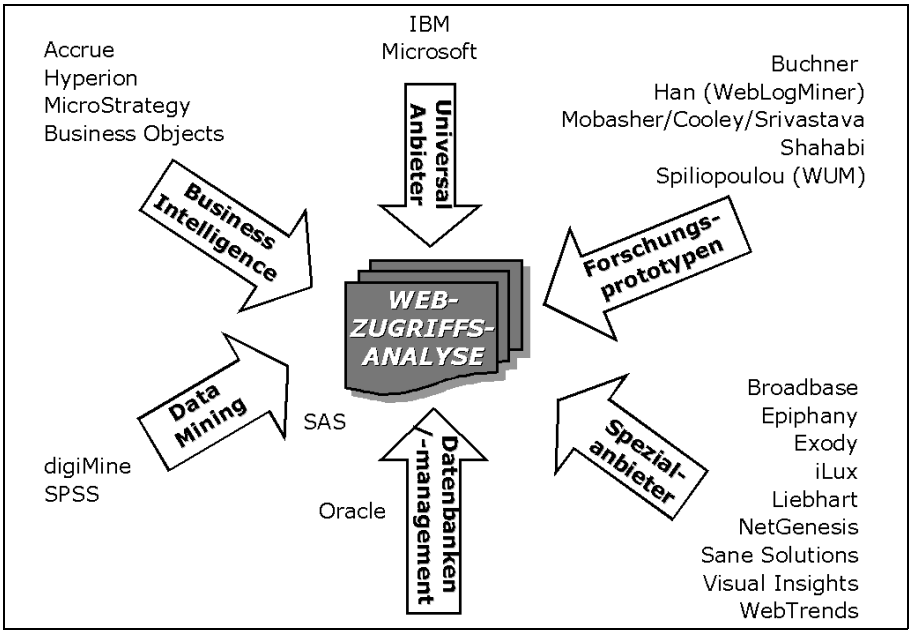


Abb. 11-9: *Anbieter von Web-Zugriffsanalyse-Produkten (Auswahl)*

Einige komplexe Werkzeuge dagegen ermöglichen eine Integration von Zugriffs- und Website-Struktur-Daten aus den Log-Dateien mehrerer Web- und Applikations-Server und Daten aus dem Unternehmens-Data-Warehouse. Zusätzlich kann eine Bewertung der Zugriffsverhalten, teilweise teil-automatisiert, über geschäftsorientierte Metriken erfolgen. Manche Produkte bieten außerdem eine Umsetzung der Ergebnisse in personenbezogene Marketing-Aktionen oder (automatische) Website-Anpassungen, inklusive einer Erfolgskontrolle der durchgeführten Maßnahmen und Verwaltung der auf einer Website angebotenen Produkte und deren Käufer.

Zur Diskussion unterteilen wir die Menge der Web-Zugriffsanalyse-Werkzeuge in drei Klassen:

- Werkzeuge für einfache Web-Zugriffsstatistiken (1)
- Data-Warehouse-basierte Werkzeuge mit OLAP-Funktionalität (2)
- Data-Mining-Werkzeuge (3)

Bei dieser Unterteilung zählt nicht nur die Art der Analysefunktionalität; vielmehr spielen Qualität, Umfang und Sicherheit der zur Verfügung stehenden Daten sowie die Fähigkeiten zur Umsetzung der erzielten Ergebnisse und deren Kontrolle eine wesentliche Rolle (vgl. Abschnitte 11.2, 11.3). Abb. 11-10 zeigt wichtige Unterkriterien wie *Art der Datenquellen*, *Datenhaltung*, *Qualität der Datentransformation*, *Art der Datenanalyse* und *Umsetzung/Kontrolle der Ergebnisse* sowie deren Ausprägungen. Als Beispiele zur Abgrenzung von Werk-

Kriterium	Ausprägungen
Art der Datenquellen	Website-Daten: Web-Server-, Browser-, Proxy-Logs, selbstdefinierte Logs (Web-Server-Plug-ins), Website-Struktur, Kundentransaktionen im Web
	Application Server Logs
	Unternehmensdaten: Kunden-/Produktinformationen, Unternehmens-Warehouse/-Data Marts
Datenhaltung	Datei(en), Datenbank, Data Warehouse
Qualität der Datentransformation	Qualität der Identifikation von: Page Views, Benutzer, Sessions, Suchbegriffe
	Filterungen: Roboterzugriffe, Dateiformate, ...
	Inhaltliche Kategorisierung von URLs
Art der Datenanalyse	tabellarische Zugriffsstatistiken, OLAP-Analyse, Data-Mining-Techniken
	(Geschäfts-)Metriken (Kaufrate, Mikrokonversionsraten, ROI)
Umsetzung/Kontrolle der Ergebnisse	Website-Anpassung
	Personalisierung
	Kampagnen-Management
	erreichter Automatisierungsgrad

Abb. 11-10: *Unterscheidungskriterien für Web-Zugriffsanalyse-Werkzeuge*

zeugen seien die emminenten Leistungs- und Verfügbarkeitsunterschiede zwischen der Datei- bzw. Datenbankhaltung der zu analysierenden Daten genannt, Aspekte der Datentransformation, bei der die schwierige, aber auch Ergebnis-kritische Bestimmung der »Besucher« und »Sessions« aus den Web-Log-Daten vorgenommen wird, oder Möglichkeiten zur Personalisierung der Auswertung.

Zur Auswahl und Planung des Einsatzes von Werkzeugen für ein Unternehmen ergeben sich zusätzliche Kriterien. Insbesondere die »*Passgenauigkeit*« des Werkzeuges in die Unternehmenslandschaft (Interoperabilität, Software-/Hardware-Gegebenheiten, Anbieter-«Politik« des Unternehmens, evtl. Outsourcing an einen ASP[1] vs. In-House-Nutzung) oder die *Marktsituation* sowohl des Produkts als auch des Anbieters sind hier zu nennen. Weiterhin sind die *Performanz* (insbesondere der Datenaufbereitung und der Anfragebearbeitung auf grossen Datenmengen), *Ergonomie* und nicht zuletzt das recht unterschiedliche *Pricing* zu nennen.

Im Folgenden konzentrieren wir uns bzgl. der Definition der Werkzeugklassen und der Abgrenzung der Werkzeuge auf die Kriterien gemäß Abb. 11-10. Wir skizzieren die wichtigsten Eigenschaften der betrachteten drei Werkzeugklassen, benennen jeweils eine Auswahl aus der umfangreichen Menge von Web-Zu-

1. ASP = Application Service Provider

griffsanalyse-Werkzeugen sowie spezielle Eigenschaften einiger Vertreter. Die Werkzeugauswahl und die abgeleiteten Aussagen orientieren sich an den verfügbaren Materialien auf den Internet-Seiten der Anbieter bzw. eigenen Erfahrungen mit den (wenigen) frei zugänglichen Demo-Versionen (Stand 4/2002). Es wurden keine Evaluierungen oder Testinstallationen unter Mitwirkung der Anbieter durchgeführt. Detaillierte Informationen finden sich unter der in Abb. 11-11 bereitgestellten Sammlung von Internet-Links zu den genannten Anbietern und Produkten.

11.7.1 Werkzeuge für einfache Web-Zugriffsstatistiken

In dieser Werkzeugklasse erfolgt die Datenhaltung üblicherweise in Dateien oder proprietären Datenbanken. Die Auswertung beschränkt sich auf (Offline-)Web-Log-Dateien, die Darstellung der Analyseergebnisse erfolgt tabellarisch oder mit rudimentären Grafiken auf der Basis fester Dimensionen (Zeit, Besuch, Nutzer). Die Datentransformation ist einfach (Ausschluss bestimmter Sites oder Dateiendungen, einfache Identifikation von Benutzern, keine inhaltliche Kategorisierung von URLs). Eine Umsetzung oder Kontrolle von Ergebnissen muss manuell erfolgen. Einfache »geschäftsorientierte« Metriken sind teilweise vorhanden, wie z.B. die Ermittlung einer Click-through-Rate als Maß für den Erfolg eines Werbebanners. Personalisierung erfolgt nicht. Vertreter dieser Kategorie sind (u.a):

- Exody WebSuxess
- IBM SurfAid for Reporting
- Liebhart systems WebFeedback
- MrUnix Webalizer
- Sane Solutions NetTracker
- WebTrends Reporting Server

Das einfachste Werkzeug dieser Gruppe stellt der *Webalizer* dar. Es produziert einen überwiegend tabellarischen Report mit allgemeinen Leistungs- und Zugriffsstatistiken (z.B. Zugriffe bzgl. Seiten, Übertragungsmenge, Besuche, fehlerhafte Seiten, zugreifende Rechnernamen etc.), deren zeitliche Entwicklung sowie Rankings (Top-Referrer/-Seiten/-Pfade/-Rechner/-Browser etc.). Sämtliche Auswertungsdimensionen sind statisch, Einstellungen werden über eine Datei vorgenommen. Quelle ist die Web-Log-Datei eines Servers.

NetTracker, *SurfAid for Reporting*, *Reporting Server* und *WebSuxess* liefern eine vorgegebene Menge von Analyseberichten, welche Zugriffsstatistiken bzgl. der Zeit, Besuche oder Pfaden umfassen. *SurfAid for Reporting*, *Reporting Server* und *WebSuxess* bieten neben Tabellenausgaben auch eine grafische Darstellung der Ergebnisse (z.B. den zeitlichen Verlauf). Eine einfache Wahl eines Abschnitts der Zeitdimension (z.B. ein bestimmte(r) Tag/Woche/Monat) ist i.d.R. möglich. Auch werden Rankings bzgl. einfacher Konversionsraten angeboten, wie man sie z.B. aus dem Verhältnis der Zugriffe über ein Werbebanner zu denen auf einer

Produktseite ablesen kann. Alle genannten Werkzeuge bedienen sich einer Datenhaltung in einer proprietären Datenbank.

IBM bietet Web-Zugriffsanalyse-Reports generell als Application Service an. *IBM* erstellt ein kundenspezifisches Warehouse und liefert Reports und/oder einen Zugang zum Data Warehouse. Die einfachste Variante und damit einen Vertreter der einfachen Kategorie stellt *SurfAid for Reporting* dar.

WebFeedback liefert eine URL-Zugriffsstatistik auf Basis der Struktur einer bis zu einer bestimmten Verzweigungstiefe zu analysierenden Website. Diese umfasst z.B. die grafische Darstellung der Verknüpfungen der URLs, die von einer Seite abgehen bzw. eine Seite erreichen, und deren Besuchshäufigkeit. Reports können in XML im- und exportiert werden.

Die Preise dieser Werkzeuge erreichen drei- bis vierstellige Euro-Beträge. Viele Anbieter stellen Demo-Versionen zur Verfügung (s.u.). Einige Werkzeuge sind sogar kostenfrei (z.B. *Webalizer*).

11.7.2 Data-Warehouse-basierte Werkzeuge mit OLAP-Analyse

Die Werkzeuge dieser Kategorie basieren auf einem Data Warehouse und meist mehreren daraus abgeleiteten Data Marts über Web-Zugriffe. Die Werkzeuge nutzen die Infrastruktur der Data-Warehouse-Lösung desselben Anbieters zur Datenaufbereitung, -haltung und -analyse; sie sind jedoch selbst zusätzlich zu erwerben.

Bezüglich der in Abb. 11-10 genannten Kriterien sind als vorherrschende Eigenschaften dieser Produktklasse zu nennen:

- Als Datenquellen fungieren neben den Log-Dateien eines oder mehrerer Web-Server häufig zusätzliche Web-Zugriffsdaten. So werden durch Web-Server-Plug-ins, »Packet Sniffer«, Browser-Plug-ins, Application Server Logs, Analyse der Website-Struktur etc. umfangreichere bzw. zeitnähere Daten gewonnen. Die Werkzeuge arbeiten auf einem Data Warehouse, welches neben den Web-Zugriffsdaten häufig auch andere Datenquellen (z.B. Kundendaten, -transaktionen, Data Marts) integrieren kann.
- Die Datentransformation erfolgt über eine grafische Oberfläche und ist üblicherweise ausgefeilter als die der einfachen Werkzeuge. So können z.B. zur Reduktion der Web-Log-Daten verfügbare Listen von Roboter-Sites integriert werden oder Suchbegriffe aus dynamischen Anfragen extrahiert werden. Eine hierarchische, inhaltliche Kategorisierung von Webseiten wird nicht unterstützt; allerdings ist es häufig möglich, Metadaten zur Beschreibung von URL(-Gruppen) zu definieren.
- Die Datenhaltung erfolgt mit gängigen (häufig allerdings ausschließlich proprietären) relationalen und/oder multidimensionalen DBS.
- Die Auswertung wird mittels Eigen- oder Fremd-OLAP-Werkzeugen mit entsprechender Roll-up/Drill-down-Funktionalität durchgeführt. Die Auswer-

tungsdimensionen sind i.d.R. vorgegeben. Ergänzt wird die Analyse durch die für einfache Web-Zugriffsanalyse-Werkzeuge typischen Statistiken und Diagramme. In der Regel können Benutzergruppen definiert werden, bzgl. derer Auswertungen durchgeführt werden.

- Die meisten Werkzeuge bieten inhaltsorientierte Metriken (z.B. Konversionsraten) an; einige unterstützen auch (personalisiertes) Kampagnen-Management.

Vertreter dieser Kategorie sind u.a:

- Accrue Hitlist/Insight
- Hyperion Web Analysis SuiteIBM SurfAid for Analysis/Business (ASP)
- Microsoft Commerce Server
- NetGenesis NetGenesis*5*
- Oracle Clickstream Intelligence
- SAS WebHound
- TeaLeaf Technologies TeaCommerce Suite

NetGenesis5 bietet eine vergleichsweise umfangreiche Menge von – auch geschäftsorientierten – Metriken zur Bewertungen des Web-Auftritts. Es können eigene Metriken und Benutzergruppen definiert werden. *NetGenesis* bietet eine zeitlich begrenzte, kennwortgeschützte Nutzung seiner Demo-Installationen (z.B. Demo-Data Mart zum Online-Buchhandel / Online-Aktienhandel). *NetGenesis5* nutzt die OLAP-Engine von *MicroStrategy*, die allerdings auch ein eigenes Web-Zugriffsanalyse-Modul *Web Traffic Analysis Module* anbieten.

Microsoft liefert mit den *Commerce Server* eine Suite, die viele Aspekte der Durchführung von Online-Shops abdeckt. Zusätzlich zur Web-Zugriffsanalyse können Webseiten generiert werden, Kunden, Produkte, Umsätze einer Website verwaltet und gezielte B2C-Kampagnen durchgeführt sowie deren Effektivität ausgewertet werden. Die Lösung beschränkt sich aber auf die Microsoft-Produktwelt (Web-Server, DBS, Analyse).

Oracle Clickstream Intelligence ermöglicht die nutzerspezifische Erweiterung des Data-Warehouse-Schemas zur Anpassung an Analyseanforderungen. Oracle liefert der immanenten Datenbanktechnologie (Parallelität, materialisierte Sichten, Datenpartitionierung) entsprechende Tuning-Möglichkeiten.

Die *TeaLeaf*-Suite bietet mittels Plug-ins für Web-Clients, -Server und einige Application Server (z.B. von SAP) die wohl weitreichendste Akquise und Darstellung von Web-Zugriffsdaten. Sie ermöglicht z.B. die Ermittlung der tatsächlichen Verweildauer von Nutzern auf Webseiten, dezidierter Fehlerursachen oder Nutzereingaben durch Browser-Plug-ins. Mittels eines Visualisierungswerkzeugs können besuchte Seiten, durchgeführte Eingaben etc. in einer Art »Slide-Show« en detail dargestellt werden. Bei der Verwendung dieses Werkzeugs sollte der Einhaltung des Datenschutzes daher besondere Beachtung geschenkt werden. Der Anbieter weist auf seine diesbezüglichen Verkaufsrichtlinien und entsprechende Software-Unterstützung (opt-in/opt-out) hin.

Bei der Auswahl einer kommerziellen Web-Zugriffsanalyse-Lösung aus dieser Werkzeugklasse sollte insbesondere die Integration in die Unternehmensproduktwelt genau beachtet werden, da die Interoperabilität mit Fremdanbietern häufig nicht oder nur beschränkt gewährleistet ist. Die (Neu-)Anschaffungskosten können für diese Werkzeuggruppe wegen der notwendigen Infrastruktur leicht im fünf- bis sechsstelligen Euro-Bereich liegen.

11.7.3 Data-Mining-Werkzeuge

Eine weitere Werkzeugklasse im Rahmen der Web-Zugriffsanalyse stellen Data-Mining-Werkzeuge dar. Insbesondere im kommerziellen Umfeld stellt Web Usage Mining im Wesentlichen ein Anwendungsgebiet für die »klassischen« Data-Mining-Verfahren zur Klassifikation, Cluster-, Assoziations- und Sequenzanalyse dar. So müssen die zu analysierenden Daten aus den vorliegenden Data Warehouses oder Dateien für die meist datei- oder tabellenorientierten Data-Mining-Werkzeuge vom Benutzer, mit größerer oder geringerer Unterstützung durch das Werkzeug, abgeleitet und aufbereitet werden. Als bekannte kommerzielle Vertreter sind u.a. zu nennen:

- digimine (ASP)
- IBM Intelligent Miner for Data
- NCR/Teradata TeraMiner
- Oracle Data Mining Suite (Darwin)
- SAS Enterprise Miner
- SPSS Clementine Workbench

Diese Werkzeuge bieten in der Regel mehrere Klassifikations-, Cluster- und Prognose-Algorithmen, die auf Web-Zugriffsdaten z.B. zur Navigationspfad-Analyse angewendet werden können. Für die typischen Web-Zugriffsanalysen offerieren die Hersteller meist zusätzliche Produkte (z.B. *SAS Webhound*, *IBM SurfAid als ASP*). SPSS Clementine allerdings bietet für sein Data Mining-Werkzeug das sog. Clementine Application Template (CAT) for Web Mining an, also eine Prozessfolge zur Aufbereitung und Analyse von Website-Besuchen bzw. -sequenzen. Daneben existiert mit CAPRI noch ein Plugin zur Sequenzerkennung. Neben der kommerziellen Produkten gibt es eine Reihe von auf Web (Usage) Mining-Verfahren spezialisierte Forschungsprototypen [SCDT00], von denen z.B. WebSIFT [CoTS99], WebLogMiner [ZaXH98] oder WUM [Spil00] zu nennen sind.

Ein generelles Manko derzeitiger Data-Mining-Werkzeuge ist, dass sie häufig auf Dateien arbeiten, selten auf (einfachen) Datenbanken und gar nicht auf Data Warehouses. Zur Vermeidung redundanter, aus den Datenbanken abgeleiteter Dateien sowie aus Leistungsgründen ist somit eine engere Anbindung der Werkzeuge an die Data Warehouses dringend geboten, um das Potenzial der Mining-Analysen voll nutzen zu können.

11.7.4 Informationen im WWW

Viele Anbieter stellen Einstiegsinformationen über ihre Web-Zugriffsanalyse-Lösungen im Internet zur Verfügung, welche erfreulicherweise häufig über die üblichen Marketing-orientierten Data Sheets hinausgehen. Abb. 11-11 zeigt eine Auswahl der Adressen von 20 der wichtigsten der über 80 kommerziellen Produkte, die während der Recherchen ausgemacht werden konnten. Die Tabelle enthält auch Verweise zu Webseiten, auf denen freie Werkzeug-Demonstrationen zur Eigeninstallation oder Direktausführung zu finden sind. Einige Anbieter allerdings liefern (mit Wissensstand 4/02) leider nur sehr wenig Online-Information über ihre wahrscheinlich interessanten Produkte wie z.B. *Elytics Analysis Suite, E.piphany E-Commerce Reporting & Analysis, iLux Enterprise CampaignManager, MicroStrategy WebTraffic Analysis Module, Quadstone Customer Conversion*.

Anbieter	Tool	Internet-Adresse	Bem. / Demos
Accrue	Hitlist/Insight	http://www.accrue.com/ Company/Contact_Us/ reports_and_white_papers.html	
digiMine	Ebusiness Analytics (ASP)	http://www.digimine.com	http://www.digimine.com/solutions/samplereports-1.asp
Elytics	Analysis Suite	http://www.elytics.com	
E.piphany	E-commerce Reporting & Analysis	http://www.epiphany.com/ products/campmgr.html	
Hyperion	Website Analysis Suite	http://www.hyperion.com/ solutions/index.cfm	Link »Online Demos«
IBM	SurfAid (ASP)	http://www-3.ibm.com/ e-business/solution/28176.html	
IBM	SpeedTracer	http://www.alphaworks.ibm.com	http://www.downloadsafari.com/Files/utilshtmlaccs/S/SpeedTracer.html
iLux Enterprise	Campaign Manager	http://www.ilux.com	http://www.ilux.com/ products/sample_reports.html
Microsoft	Commerce Server	http://www.eu.microsoft.com/ germany/produkte/ overview.asp?siteid=10733	http://www.microsoft.com/ downloads (Microsoft download-Center)
MicroStrategy	Web Traffic Analysis Module	http://www.microstrategy.com/ Software/Applications/WTAM	
Mr Unix	Webalizer	http://www.mrunix.net/webalizer	
NCR Teradata	Teraminer	http://www.ncr.com/products/ software/teradata_mining.htm	
net.Genesis	NetGenesis5	http://www.netgen.com (download von White Papers)	

Abb. 11-11: *Auswahl von Web-Zugriffsanalyse-Tools im WWW (Stand 4/02)*

Anbieter	Tool	Internet-Adresse	Bem. / Demos
Oracle	Intelligence 1.0	http://otn.oracle.com/products/clickstream/htdocs/ocsi_faq.htm http://otn.oracle.com/products/clickstream/content.html	
Quadstone	Customer Conversion	http://www.quadstone.com	
Sane Solutions	NetTracker	http://www.sane.com/products/NetTracker/	http://www.sane.com/demo/de/NetTracker/web/index.html
SAS Institute	Web Hound, Enterprise Miner	http://www.sas.com/products/webhound/index.html http://www.sas.com/service/library/onlinedoc/webhound (Doku)	
SPSS	Clementine	http://www.spss.com/spssbi/clementine/CATs.htm http://www.spss.com/spssbi/capri/index.htm	
TeaLeaf Technology	TeaCommerce Suite	http://www.tealeaf.com/solutions/tcsuite.asp	
WebTrends	Reportings Server, Commerce Trends	http://www.webtrends.com/products/wrc/ent.htm	www.upenn.edu/computing/web/webteam/webtrends/cmplte.htm

Abb. 11-11 *(Fortsetzung): Auswahl von Web-Zugriffsanalyse-Tools im WWW (Stand 4/02)*

Neben den Produktinformationen der einzelnen Anbieter existieren zahlreiche Websites mit Informationen und Listen von Web-Zugriffsanalyse-Werkzeugen. Dort sind Verweise zu den Anbietern zu finden sowie – meist rudimentäre – Werkzeugbeschreibungen und einige wenige Werkzeugvergleiche. Beispiele sind (Stand 4/2002):

- ECRMGUIDE.COM:
  www.ecrmguide.com
- INTELLIGENT ENTERPRISE:
  www.intelligententerprise.com/000929/b_guide.shtml
  www.intelligententerprise.com/000929/feat5.shtml
- INTERNET PRODUCT WATCH: ipw.internet.com/analysis/recent1.html
- KDCENTRAL.COM:
  www.kdcentral.com/Software/Data_Mining/Web_Log_Mining
- KDNUGGETS: www.kdnuggets.com/software/web.html
- ZDNET: xlink.zdnet.com (Suchanfrage »web mining tool«)
- YAHOO:
  http://dir.yahoo.com/Computers_and_Internet/Software/Internet/World_Wide_Web/Servers/Log_Analysis_Tools

11.8 Zusammenfassung

Die Nutzungsmöglichkeiten der Web-Zugriffsanalyse sind sehr vielfältig und nicht nur für kommerzielle Websites von immer größerer Wichtigkeit, um eine gute Bedienung der Nutzer und die Erfüllung eigener Zielsetzungen für den Web-Auftritt zu erreichen. Derzeit herrschen noch einfache statistische Auswertungen auf Basis der Web-Log-Dateien vor, die jedoch viele Optimierungsmöglichkeiten ungenutzt lassen. Wir haben gezeigt, dass die Verwendung eines Data-Warehouse-basierten Ansatzes wesentliche Vorteile bringt, insbesondere hohe Skalierbarkeit, einfache Erweiterbarkeit und flexible Auswertungsmöglichkeiten. Von Bedeutung hierzu ist vor allem die Integration von Informationen zu Nutzern/Kunden und Inhalten/Produkten, um zu aussagekräftigen Ergebnissen und konkreten Verbesserungsmöglichkeiten zu gelangen.

Die Umsetzung eines Data-Warehouse-Ansatzes sowie die Nutzung von Data-Mining-Werkzeugen verursacht derzeit noch einen hohen Aufwand. Dieser sollte aufgrund der Weiterentwicklung entsprechender Werkzeuge, auch im Rahmen von kommerziellen Datenbanksystemen, zunehmend reduziert werden können. Auch aus Sicht der Forschung gibt es noch interessante, unzureichend untersuchte Fragestellungen, z.B. bei der Datenaufbereitung, der Ausgestaltung spezifischer Data-Mining-Verfahren sowie der automatisierten Umsetzung der Ergebnisse (dynamische Website-Modifikationen, Personalisierung).

Literatur

[BaGÜ01] Bauer, A., Günzel, H.: Data-Warehouse-Systeme. dpunkt.verlag 2001.

[CoMS99] Cooley, R., Mobasher, B., Srivastava, J.: *Data Preparation for Mining World Wide Web Browsing Patterns*. Knowledge and Informations Systems 1 (1), Springer-Verlag, 1999.

[CoTS99] Cooley, R., Tan, P.-N., Srivastava, J.: *WebSIFT: The Web Site Information Filter System*. Proc. Web Usage Analysis and User Profiling Workshop (WEBKDD'99), San Diego, CA, August 1999.

[EsSa00] Ester, M., Sander, J.: *Knowledge Discovery in Databases: Techniken und Anwendungen*. Springer-Verlag, 2000.

[HaKa00] Han, J., Kamber, M.: *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.

[KiMe00] Kimball, R., Merz, R.: *The Data Webhouse Toolkit: Building the WebEnabled Data Warehouse*. Wiley 2000.

[Schr00] Schroeck, M.J.: *E-Analytics – The Next Generation of Data Warehousing*. DM Review, Aug. 2000, http://www.dmreview.com.

[SCDT00] Srivastava, J., Cooley, R., Deshpande, M., Tan, P.: *Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data*. SIGKDD Explorations 1(2), 12-23, 2000.

[Spil00] Spiliopoulou. M.: *Web Usage Mining for Web Site Evaluation*. Comm. ACM 43 (8), 127-134, 2000.

[ScNL01] Schaarschmidt, R., Nowitzky, J., Lufter, J.: *Clickstream Warehousing für e-CRM: Neue Herausforderungen an die Datenhaltung?* Proc. 5. Wirtschaftsinformatik (WI)-Tagung, pp. 117-131, Augsburg, Sep. 2001.

[StRQ00] Stöhr, T., Rahm, E., Quitzsch, S.: *OLAP-Auswertung von Web-Zugriffen*. Proc. GI-Workshop Internet-Datenbanken, Sep. 2000, http://dol.uni-leipzig.de/pub/2000-23

[WKDD01] WEBKDD-Workshops zu Web Mining: http://robotics.Stanford.EDU/~ronnyk/WEBKDD2000 bzw. WEBKDD2001.

[ZaXH98] Zaiane, O. R., Xin, M., Han, J.: *Discovering Web Access Patterns and Trends by Applying Olap and Data Mining technology on Web Logs*. Advances in Digital Libraries, pp. 19-29, Santa Barbara, CA, 1998.

12 Web-basiertes Lernen: Eine Übersicht über Stand und Entwicklungen

Peter Jaeschke, Andreas Oberweis, Gottfried Vossen

Kurzfassung

Web-basiertes Lernen, häufiger inzwischen auch unter dem Begriff E-Learning subsumiert, wird heute in Ausbildungsstätten sowie in Unternehmen intensiv diskutiert und an vielen Stellen sowie in zahlreichen Anwendungen bereits praktiziert. Man versteht darunter einerseits die Verlagerung von Lernsituationen, einzelnen Lektionen oder sogar vollständigen Kursen, Vorlesungen oder Seminaren zur Aus- oder Weiterbildung unterschiedlichster Länge, Natur und Zielsetzung auf elektronische Medien, insbesondere solche, die sich des Internet und des Web bedienen, und andererseits eine zeitliche, räumliche und inhaltliche Flexibilisierung bzw. Personalisierung der Lernprozesse. Man möchte sich dadurch das Medium Web in neuartiger Weise zunutze machen und erhofft sich grundsätzlich eine Verbesserung der Vermittlung von Wissen, des Lernens und des Lehrens um Aspekte wie zeitliche, räumliche oder organisatorische Flexibilität oder Individualisierbarkeit und verbesserte Möglichkeiten des Eingehens auf die Fähigkeiten und Bedürfnisse des einzelnen Lernenden. Derzeit laufen sowohl in öffentlichen Einrichtungen wie in der privaten Wirtschaft eine Vielzahl von Aktivitäten in dieser Richtung, von denen zahlreiche zum Ziel haben, klassische Unterrichtsformen (auch) multimedial zu gestalten oder durch neue Medien teilweise oder vollständig zu ersetzen. Dieses Kapitel will Anforderungen an E-Learning-Systeme zusammenstellen, relevante Konzepte aus inhaltlicher, organisatorischer und technischer Sicht diskutieren und über exemplarische Realisierungen berichten. Wir beenden unsere Ausführungen mit dem Versuch einer Prognose, wie sich das Gebiet in naher Zukunft entwickeln wird und welche Perspektiven sich eröffnen.

12.1 Einführung

Die Unterstützung von Wissensvermittlung und Lernen, aber auch von Lehren durch so genannte »neue Medien«, also weg vom reinem Lernen nach Büchern, im Frontalunterricht sowie mit Papier und Stift, hat in den vergangenen 15 Jahren einen enormen Aufschwung genommen. Man hat nämlich erkannt, dass sich nahezu jede Form von Unterricht oder Lehre durch eine Computer-Unterstützung attraktiver und vielfältiger gestalten lässt, so dass neue Anreize geschaffen werden können, sich sogar mit vergleichsweise trockenem Stoff zu befassen. Aus dieser Erkenntnis und auf Grund einer zunehmenden Verbreitung von Rechnern auch

z.B. in Privathaushalten in Verbindung mit immer besseren Internet-Anschlüssen sind in den letzten Jahren an Hochschulen, aber auch in der Industrie zahlreiche Projekte und Produkte zum Thema E-Learning entstanden.

Als ein typisches Beispiel erwähnen wir die amerikanischen *Traffic Schools*: Wer in der Vergangenheit im amerikanischen Straßenverkehr bei einem Regelverstoß ertappt wurde, konnte zur Teilnahme an einem Wiederauffrischungskurs über Verkehrsregeln verpflichtet werden. Da diese Kurse nicht selten am Wochenende stattfanden und man zum Verbleib für die gesamte Dauer verpflichtet war, war die Motivation der Teilnehmer allenfalls darin begründet, dass ohne Teilnahmezertifikat der Führerschein einbehalten wurde. Inzwischen werden derartige Kurse im Web zum Selbststudium angeboten; man ist in der zeitlichen Gestaltung flexibel und kann z.B. den erforderlichen Abschlusstest ablegen, wann immer man sich für angemessen vorbereitet hält, so dass der Zertifikatserwerb jetzt erheblich schneller erfolgen kann.

Hinsichtlich des Web-basierten Lernens ist eine Reihe von Unterscheidungen angebracht, auf die wir als Erstes eingehen wollen.

12.1.1 Begriffliche Unterscheidungen

Zahlreiche Begriffe werden heute vereinfachend unter *E-Learning* subsumiert:

- *Computer Based Training*, CBT, bezeichnet lediglich eine Form von Training, bei der ein Computer eingesetzt wird. Lernmaterial auf CD-ROM, das sich Lernende auf dem eigenen PC installieren können, gehört somit zu CBT.
- *Computer unterstützte Lehre,* CUL, bzw. *Computer Aided Teaching,* CAT bezeichnet die (meist universitäre) Lehre unter Einsatz des Computers, wobei die Interpretationen von der Slide-Show auf einem Laptop über den mit Rechnern gefüllten Klassenraum bis zum Electronic White Board reichen.
- *Distance Learning* sowie *Teleteaching* bezeichnen das Lehren und Lernen über eine räumliche Entfernung hinweg, also z.B. vom häuslichen Arbeitsplatz bzw. von der Wohnung aus oder im Rahmen von mobilem Rechnen.
- *Web Based Training,* kurz WBT, bezeichnet das hier eigentlich im Vordergrund stehende Lernen, welches als Kommunikationsmedium das Internet verwendet und seitens des Teilnehmers im Allgemeinen nicht mehr als einen Browser erfordert. Im Unterschied zum Distance Learning findet beim WBT im Allgemeinen eine Live-Interaktion zwischen Lehrenden und Lernenden statt.

Zu WBT sei bereits an dieser Stelle bemerkt, dass allein die Verfügbarkeit eines Browsers bei kommerzieller Lernsoftware häufig nicht ausreicht, sondern dass proprietäre Software installiert werden muss, über welche dann per Web Lernsitzungen abgewickelt werden können. (Hintergründe hierfür sind z.B. abrechnungstechnischer Natur oder der Zweck einer Verfolgung der Aktivitäten des Lerners durch einen Betreuer.)

Neben diesen Unterscheidungen gibt es weitere, welche zu den genannten orthogonal sind:

- Man spricht von *synchronem* Lernen, wenn mehrere Teilnehmer eines Kurses oder einer Veranstaltung gleichzeitig auf derselben Lernplattform »versammelt« sind bzw. mit einer solchen in Kontakt stehen (diese Teilnehmer können sich physisch jedoch an unterschiedlichen Orten befinden). Das Lernen vollzieht sich dann im Allgemeinen unter der Kontrolle eines Tutors bzw. eines Lehrenden.
- Man spricht dagegen von *asynchronem* Lernen, wenn der einzelne Teilnehmer eines Kurses zeitlich unabhängig von (eventuell aktuell nicht vorhandenen) anderen Teilnehmern Lerneinheiten bearbeitet und z.B. mit einem Tutor über E-Mail oder nur zu bestimmten Zeiten über einen Chat-Room in Verbindung tritt.

Allgemeine Einführungen in unterschiedliche Facetten des E-Learning sowie des Web-basierten Lernens geben [Ross02], [AdCP02] oder [Ditt02], in universitäre Entwicklungen der gerade geschilderten Art [Dobe00], [FeSc01], [Grob00] oder [Grob01].

12.1.2 Anwendungsszenarien

Hochschulen, aber auch Schulen, die mit *Primärausbildung* befasst sind, experimentieren naturgemäß intensiv mit neuen Konzepten und erproben neue Möglichkeiten der Organisation des Lehrbetriebs, der Kooperation zwischen Individuen oder Einrichtungen oder des Austauschs von Lerninhalten über Campusgrenzen hinaus. Auch für Unternehmen, die Ausbildungsberufe anbieten, sind elektronische Kurse attraktiv, sofern sie gewissen Rahmenbedingungen genügen (siehe unten, z.B. Tutorunterstützung, interaktive Betreuung u.a.).

In der *Sekundärausbildung*, also der universitären, postgradualen, von Gesellschaften wie der IEEE getragenen [ieee] oder der innerbetrieblichen *Weiterbildung* oder auch der nebenberuflichen Fortbildung, eröffnet das Web-basierte Lernen ebenfalls neue Möglichkeiten. Insbesondere kann die Teilnahme an einem Kurs oder einer Fortbildung zeitlich und räumlich unabhängig und auf den Einzelnen abgestimmt gestaltet werden. Dies ist wesentlich, da ein derartiges Lernen im Allgemeinen neben der Ausübung eines Berufes durchgeführt wird, so dass man Anwesenheit an einem bestimmten Ort zu einer bestimmten Zeit nicht garantieren kann oder will. An dieser Stelle setzen inzwischen auch zahlreiche Gesellschaften sowie kommerzielle Anbieter an, die in der Vergangenheit zum Teil Präsenzseminare angeboten haben. Man kann eine Vielzahl von Seminarinhalten mittlerweile im Web buchen und dann selbstständig abrufen und bearbeiten, sodann Tests absolvieren und Zertifikate erwerben; als Beispiel verweisen wir auf [ieee].

Eine unmittelbare Folge der Zeit- und Ortsunabhängigkeit ist, dass sich E-Learning gut für das immer wichtiger werdende (und nicht selten lebenslang betriebene) *Lernen bei Bedarf* (Learning on Demand) einsetzen lässt. Dabei werden Inhalte nicht notwendig in Kursen, sondern in kleinen und kleinsten, teilweise standardisierten oder individualisierten Einheiten vermittelt. Das Abrufen von Inhalten muss spontan und auf die jeweilige Situation bezogen erfolgen können, und es muss sich im Idealfall in den laufenden Geschäftsbetrieb integrieren lassen. Insbesondere Firmen setzen diese Form des Lernens zur Ausbildung von Mitarbeitern ein, zur Schulung im Hinblick auf neue Produkte oder etwa zur Weitergabe neuer Richtlinien. Als Beispiel denke man an die Mechaniker einer Werkstatt, die sich in die Besonderheiten eines neuen Fahrzeugmodells einarbeiten müssen; man denke ferner an die Schulung von Mitarbeitern in Richtung von betrieblichen Prozessen, etwa in Marketing oder Vertrieb (wie bringe ich ein Produkt an den Käufer?), in der Herstellung, im Accounting und Controlling, in der Verwaltung, im Call-Center. Nach [EfHo98] wird das bisherige »Lernen auf Vorrat« in Zukunft immer stärker durch Lernen bei Bedarf oder auf Anforderung ersetzt werden. Schließlich kann sich das elektronische Lernen auch auf so genannte »Soft Skills« beziehen wie Problemlösen oder Mitarbeiterführung. In allen gerade genannten Fällen wird Web-basiertes Lernen häufig durch Intranet-basiertes Lernen ersetzt, wobei grundsätzlich die gleichen Techniken und Produkte angewendet bzw. eingesetzt werden.

Wir wollen hier als Arbeitsdefinition von E-Learning die folgende verwenden: *E-Learning bezeichnet orts- und zeitunabhängiges, bei Gruppen ferner verteiltes, weitgehend selbst organisiertes und selbst gesteuertes Lernen und Erarbeiten von Wissen im Web oder mit Web-basierten Techniken anhand vorgegebener und adaptierbarer Lernpläne, Navigationsstrukturen und inhaltlicher Komponenten zum Zwecke des Erreichens eines festgelegten Lernziels.*

Da es beim Lehren und Lernen grundsätzlich um die Vermittlung von Fertigkeiten und von *Wissen* geht, wird der Bereich E-Learning nicht selten auch noch von einer anderen Seite her betrachtet. Speziell im Kontext eines Unternehmens, welches E-Learning zur firmenspezifischen oder firmeninternen Aus- sowie Weiterbildung einsetzt, will man häufig den Mitarbeitern nicht nur Wissen *vermitteln*, sondern mittelfristig auch ihr angeeignetes Wissen und Erfahrungen für das Unternehmen *konservieren*; zum reinen Lernen kommt dann der Aspekt des Wissensmanagement hinzu. Davon erhofft man sich zum Beispiel, dass man neuen Firmenmitarbeitern über Techniken des E-Learning den Wissensstand eines Unternehmens beibringt und umgekehrt über Feedback und Dokumentation von erarbeitetem Mitarbeiterwissen das »Corporate Knowledge« erhält bzw. erweitert.

12.1.3 Aufbau von Lernsystemen

Unabhängig davon, ob E-Learning in der Primär-, Sekundär- oder laufenden Aus- und Weiterbildung eingesetzt wird, lassen sich stets bestimmte *Akteure* sowie

Systemkomponenten identifizieren, welche im Rahmen einer Lernplattform bzw. eines Lernsystems zusammenspielen, vgl. [IMS01]:

- Es sind Personen in unterschiedlichen Rollen beteiligt: Als *Autoren* sind sie für die Erstellung von Inhalten zuständig, als *Tutoren* für deren Betreuung im laufenden Lehrbetrieb, als *Administratoren* verwalten sie Inhalte und Informationen aus technischer, als *Sachbearbeiter* aus organisatorischer Sicht und als *Lernende* nutzen sie das System. Autoren lassen sich weiter untergliedern in Abhängigkeit davon, für welche Art von Inhalt sie verantwortlich sind, z.B. Kurserstellung oder Erstellung eines Tests. Natürlich kann ein und dieselbe Person je nach Anwendungssituation mehrere Rollen gleichzeitig innehaben.
- Systemseitig benötigt man *Autorensysteme*, also Systeme zur Erstellung und zum Austausch von Inhalten, sodann benötigt man Systeme zur Integration und zur Speicherung von Inhalten, die von eventuell unterschiedlichen Autoren erstellt wurden, sowie zur Verwaltung von Informationen über Inhalte und über Lernende, welche heute im Allgemeinen als *Learning Management Systems* (LMS) bezeichnet werden, und man benötigt *Laufzeitsysteme*, über welche Lernende mit der Inhaltsdatenbank und dem LMS interagieren.

Dieser allgemeine Aufbau eines Lernsystems, von dem sich im Einzelfall natürlich Abweichungen ergeben können, ist in Abbildung 12-1 gezeigt.

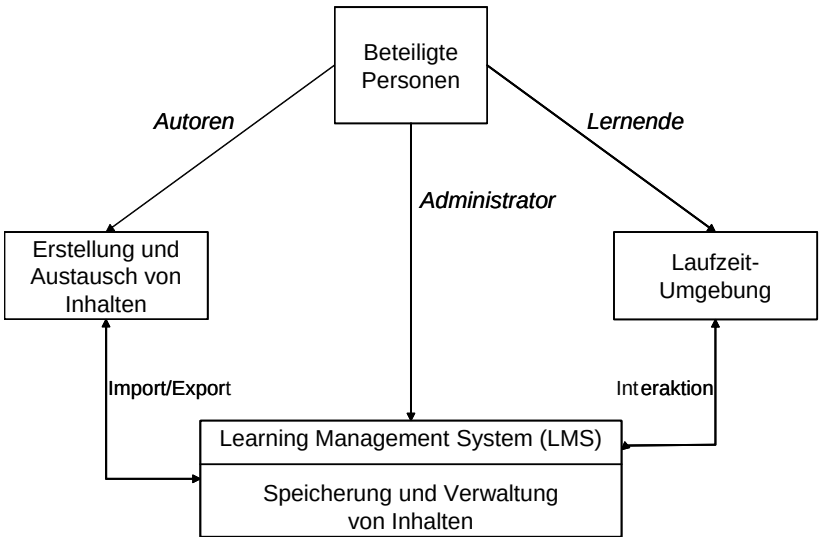


Abb. 12-1: *Komponenten und Beteiligte eines Lernsystems.*

Wir wollen die in Abbildung 12-1 gezeigte Unterscheidung in diesem Kapitel durchgehend verwenden bzw. des Öfteren auf sie zurückkommen, denn für alle drei der gezeigten Systemkomponenten ergeben sich unterschiedliche Anforde-

rungen und Realisierungsmöglichkeiten. Bei Autorensystemen geht es z.B. um die effiziente Erstellung von Inhalten; da dies ein aufwändiger Prozess ist, der zahlreiche Aspekte zu berücksichtigen hat (technischer, mediendidaktischer oder auch lernpsychologischer Natur), besteht ein großes Interesse an einer Wiederverwendbarkeit einmal erstellter Inhalte. Inhalte, auf die im Rahmen einer Lernplattform zurückgegriffen werden kann, werden nicht selten von mehr als einem Autor erstellt bzw. aus unterschiedlichen Quellen zusammengestellt. Nun weiß der Informatiker aus dem Software Engineering, dass *Reuse* nur dann erzielbar ist, wenn man sich auf eine angemessene Modellierung sowie Dokumentation und die weitgehende Einhaltung von Standards verlassen kann. Dementsprechend spricht man auch im Kontext von Autorensystemen bereits von *Lernobjekten*, aus denen Kurse aufgebaut werden, und es sind bereits *Standards* in der Diskussion, die auf deren Austauschbarkeit zwischen unterschiedlichen Autorensystemen abzielen. Hiervon wird noch genauer die Rede sein.

Autorensysteme, LMS und auch die Laufzeitumgebungen von Lernplattformen verwenden Datenbanken als zentrale Speicher- und Organisationsplattformen für Inhalte sowie für administrative Daten, und die bereits skizzierten Anwendungen stellen an solche Datenbanken zahlreiche Non-Standard-Anforderungen wie hohe Verfügbarkeit, Verwaltung multimedialer sowie benutzerdefinierter Daten, Versionsunterstützung für Inhalte, hohe Sicherheitsanforderungen bei der Verwaltung von Prüfungsinformationen über das Web, effiziente Suchfunktionalität, Skalierbarkeit u.a.

Wie das oben genannte Beispiel der amerikanischen Traffic Schools andeutet, muss sich die Laufzeitumgebung eines Lernsystems an die Lernenden anpassen können: Wer »lediglich« zu schnell gefahren, ansonsten mit den Verkehrsregeln jedoch vertraut ist, kann die geforderte Prüfung wahrscheinlich eher ablegen als jemand, der die Bedeutung bestimmter Verkehrsschilder nicht kennt. An solchen Stellen kommen *Adaptionstechniken* zum Einsatz, die den Verlauf eines Kurses an die Vorkenntnisse sowie den Lernfortschritt des Einzelnen anpassen und darüber z.B. steuern, ab wann oder wie oft in den Prüfungsbereich eines Kurses verzweigt werden darf. Eine einfache Form der Adaption ist z.B. über *Hypertexte* erzielbar, also Texte, in denen über Links auf andere Texte verzweigt wird, so dass man z.B. Erläuterungen zu einem in einem Text vorkommenden Begriff per Mausklick erreichen kann. Ein weiterer wichtiger Aspekt einer Laufzeitumgebung ist, dass sie gegebenenfalls die Gegenwart einer Lerngruppe, etwa durch eine entsprechende Anzeige am Bildschirm, deutlich macht, so dass Lernende miteinander in Kontakt treten können.

Da man sowohl beim Erstellen, Zusammensetzen und Verteilen von Inhalten als auch und gerade beim Lernen selbst von *Prozessen* redet, kann man sich als angemessene unterstützende Technik eines Lernsystems (de facto in allen drei Teilsystemen) auch ein Workflow-Managementsystem vorstellen. Mit diesem werden die vorkommenden Prozesse modelliert, etwa eine Vorgehensweise bei der Erstellung eines Kurses oder ein Vorschlag für eine Bearbeitungsreihenfolge von Kurseinheiten; das System kann sodann deren Ausführung überwachen, be-

nötigte Werkzeuge zum richtigen Zeitpunkt starten oder Fortschritt in einer Kursbearbeitung bewerten. Wichtige Voraussetzung für einen effektiven Einsatz ist eine gewisse Flexibilität, die es ermöglicht, von einem vordefinierten Prozess-Schema im Einzelfall auch kontrolliert abweichen zu können. Es ist sogar vorstellbar, dass der Umgang mit Inhalten und Kursen in naher Zukunft als ein *Web Service* zur Verfügung steht, wie er bisher für Anwendungen z.B. im Electronic Commerce konzipiert wurde: Ein Lernwilliger kann im Web zunächst nach für ihn passenden Inhalten suchen, geeignete sodann buchen und gebuchte schließlich über ein LMS bearbeiten. Eine Service-Plattform leistet die in einem solchen Kontext benötigten Austauschprotokolle, Kommunikations-, Konfigurations- und Abrechnungsmechanismen.

12.1.4 Weiteres Vorgehen

Im weiteren Verlauf dieses Kapitels wollen wir zunächst einen Anforderungskatalog für Web-basierte Lernsysteme aufstellen. Dabei ergeben sich unterschiedliche Dimensionen in Abhängigkeit von den Zielen, die man im konkreten Anwendungsfall mit E-Learning verfolgen will. Sodann wollen wir konzeptionelle Aspekte (z.B. Benutzermodellierung oder Standardisierung) sowie methodische und technische Grundlagen des E-Learning durchleuchten. Schließlich werden wir einige Beispielsysteme beschreiben. Dies stellt auch eine Verbindung zu anderen Kapiteln dieses Buches her. Den Abschluss der Ausführungen bildet ein Ausblick auf zu erwartende Entwicklungen im Bereich des E-Learning.

12.2 Anforderungen an E-Learning-Systeme

In diesem Abschnitt wollen wir Anforderungen an elektronische Lernsysteme formulieren, die sich aus den Einsatzbereichen für diese Systeme in Schulen und Hochschulen sowie im betrieblichen Umfeld ergeben. Dabei behandeln wir zunächst allgemeine Anforderungen und werden anschließend konkreter anhand der in Abbildung 12-1 gezeigten Einteilung, d.h., wir behandeln inhaltliche Anforderungen, die sich vor allem an Autorensysteme sowie an LMS richten, sodann organisatorische und technische Anforderungen primär an Laufzeitumgebungen.

12.2.1 Allgemeine Anforderungen

Wesentliche allgemeine Anforderungen an ein E-Learning-System, insbesondere an die darin verwendeten Clients, sind *Ortsunabhängigkeit*, *Zeitunabhängigkeit* und *Unterstützung von Mobilität*. Den Lernenden muss es möglich sein, an wechselnden Orten (zu Hause, am Arbeitsplatz, im Internet-Cafe, unterwegs) zu von ihnen frei wählbaren Zeiten mit dem System zu arbeiten (was für das betreffende System dann eine »24/7-Verfügbarkeit« erfordert). Im Idealfall ist zum Zugriff auf das System lediglich ein gängiger Web-Browser erforderlich, so dass von jedem

Rechner, auf welchem ein solcher installiert ist, Lerninhalte abgerufen und bearbeitet werden können. Dies sichert Ortsunabhängigkeit und unterstützt die Mobilität der Teilnehmer, da Browser heute zu den gängigen Ausstattungsmerkmalen eines jeden Rechners gehören, der entweder direkt an das Internet angeschlossen ist oder von dem aus man sich in das Internet einwählen kann.

Mobilität bedeutet allerdings nicht nur, dass ein Systemnutzer seinen Standort wechseln kann, sondern dies bedeutet im Idealfall, dass ein Benutzer Teile seiner Applikationen oder seiner Daten auf ein lokales System herunterladen kann, dort verändern und zu einem späteren Zeitpunkt mit dem zentralen Server wieder synchronisieren kann. Angewandt auf eine Lernsituation kann das z.B. bedeuten, dass man sich eine Lerneinheit sowie einen zugeordneten Test aus dem Netz auf einen Laptop lädt, die Lerneinheit bearbeitet und den Test macht, während man nicht eingeloggt ist, und nach Abschluss den Test zum Lernserver schickt. Aus dem Wunsch nach derartigen Formen von Mobilität ergeben sich technische Anforderungen, von denen weiter unten noch zu reden sein wird, wobei streng genommen zu unterscheiden ist zwischen Mobilität im lokalen Bereich, etwa zwischen den Gebäuden auf einem Universitätscampus, zwischen denen ein Funknetz existiert, und Mobilität über größere geographische Distanzen, etwa während einer Zugfahrt, während der man über GPRS mit einem Server in Verbindung steht.

Es sei bemerkt, dass es je nach Anwendungsbereich Anforderungen gibt, welche sich in die nachfolgenden Unterabschnitte nicht oder nur unzureichend einordnen lassen. Wir denken hier zum Beispiel an Anforderungen, welche sich im Rahmen des Erlernens kooperativer Arbeitsprozesse, vgl. [Haak01], ergeben.

12.2.2 Inhaltliche Anforderungen

Bei *inhaltlichen* Anforderungen ist, wie bereits erwähnt, zu unterscheiden zwischen Anforderungen an ein Autorensystem, mit welchem Inhalte erstellt werden, Anforderungen an die Inhalte selbst, Anforderungen an ein Learning Management System, welches Inhalte integriert und zentral verwaltet, sowie gegebenenfalls Anforderungen aus der Sicht von Wissensmanagement. Inhaltliche Anforderungen richten sich ferner nach der Zielgruppe bzw. den Adressaten, für die ein Lernsystem eingerichtet wird, denn es ist klar, dass Lernsysteme für Ausbildungszwecke vollständige, didaktisch aufbereitete Kurse anbieten müssen, wohingegen man bei betrieblichen Anwendungen z.B. auf den Bezug zum Betrieb oder die möglichst direkte Umsetzbarkeit der Lerninhalte Wert legt (oder auch auf die Tatsache, dass zum Lernen keine oder höchstens unregelmäßig feste Zeiten zur Verfügung stehen).

Bei der Erstellung von elektronisch aufbereiteten Lerninhalten ist bisher meist ein hoher zeitlicher und finanzieller Aufwand zu treiben, der nicht selten gescheut wird und der allenfalls dann gerechtfertigt ist, wenn die fertigen Inhalte zumindest wieder verwendet werden können. Autorensysteme sind daher mit der zentralen Anforderung konfrontiert, den Erstellungsprozess von Lerninhalten zu un-

terstützen bzw. zu vereinfachen. Dies wird z.B. dann erreicht, wenn dem Autor für bestimmte Lerninhalte eine Menüführung angeboten wird, die auf einer zuvor festgelegten festen oder zumindest in Teilen variablen Strukturierung basiert. Wir werden in Abschnitt 12.3 sehen, wie derartige Strukturierungen aussehen können, und in Abschnitt 12.4 darauf eingehen, was Autorensysteme dann bieten können.

Inhaltliche Anforderungen an die Vorlesungen, Kurse, Vorträge, Seminare, Übungen, Lernmodule, Lerneinheiten, Lernobjekte oder andere Komponenten eines E-Learning-Systems ergeben sich nicht selten »von außen«, also durch einen Lehrplan, einen Ausbildungsplan oder ein Lernziel, das man mit einer bestimmten Einheit erreichen will. Als wesentlich hat sich bereits in der Vergangenheit eine modulare Organisation und Strukturierung von größeren Einheiten in Form von »Wissensbausteinen« oder »Lernobjekten« erwiesen, welche

- sich zu größeren Modulen zusammensetzen lassen,
- an die Lernenden anpassbar sind,
- unterschiedliche Lernstile unterstützen und
- auf unterschiedliche Zielgruppen ausgerichtet werden können.

Man vergleiche hierzu z.B. [LOT]. Dies bedeutet, dass man einen Wissensbaustein, ein Lernmodul oder ein Lernobjekt in unterschiedlichen Detaillierungsgraden (mit viel oder wenig Text, mit oder ohne multimediale Inhalte) entwickeln und vorhalten muss. Die jeweiligen Inhalte bedienen sich dann einer Vielzahl unterschiedlicher Dokumententypen, und die Plattform muss in der Lage sein, entsprechende Dokumente bzw. Daten und Dateien zu verwalten und gegebenenfalls abzuspielen. Auf der Seite einer Modellierung von Wissensbausteinen kann dies z.B. auf eine Objektsicht führen, in welcher jeder einzelne Baustein als ein Objekt repräsentiert wird, für das jedoch unterschiedliche Darstellungsformen, Detaillierungsgrade und Versionen erfasst werden können. Ein und dasselbe Objekt kann z.B. stichwortartig, in Kurztexten oder in einem längeren Text beschrieben sein, es kann animiert oder nur textuell dargestellt sein, oder es kann in unterschiedlichen farblichen Ausgestaltungen vorhanden sein.

Liegt ein Objekt oder Wissensbaustein in derart unterschiedlichen Versionen vor, so kann man einerseits auf unterschiedliche Anforderungen durch die Lernenden angemessen reagieren: Darstellungen sind personalisierbar; der Umfang eines Bausteins kann den Vorkenntnissen, der Relevanz des Bausteins im Kurskontext oder dem Tempo des Lernfortschritts angepasst werden. Andererseits lassen sich Objekte aus technischer Sicht vielfältig handhaben: Sie können per Aggregation zu anderen Objekten zusammengefasst werden (aus einzelnen Wissensbausteinen entsteht auf diese Weise ein Kurs oder ein Teil eines Kurses), sie können als Spezialisierung anderer Objekte aufgefasst werden, und sie können als Dokumente in eine Ablaufsteuerung oder einen Workflow einfließen, welche bzw. welcher einem Lernprozess insgesamt zugrunde gelegt wird.

Für ein Autorensystem (und ebenso für ein LMS) folgen hieraus zahlreiche weitere Anforderungen, etwa die Unterstützung multipler Datentypen (Text-,

Bild-, Ton- oder Filmdokumente), Versionsunterstützung oder die Möglichkeit der Konfigurierung von Kursen oder Kurseinheiten aus vorhandenen Dokumenten bzw. Objekten, und dies relativ zu Profilen von Lernenden, die sich im Laufe der Zeit verändern können.

Die Erstellung von Inhalten ist damit offensichtlich ein komplexer Prozess, und so verwundert es nicht, dass Inhaltserstellungsaufgaben im Allgemeinen nicht von einer einzelnen Person durchgeführt werden. Wissensbausteine oder Lernobjekte werden also selten »aus einer Hand« entwickelt, sondern von mehreren unabhängigen Autoren oder von einer kooperierenden Autorengruppe. So wird ermöglicht, dass man Kurse oder Lerneinheiten aus Bausteinen zusammensetzen kann, die von unterschiedlichen Autoren und mit ursprünglich unterschiedlichen Zielen entwickelt wurden. Dies ist durchaus noch analog zu der Idee von einer (konventionellen) Vorlesung, die sich unterschiedlicher Quellen bedient (Kapitel aus verschiedenen Lehrbüchern, daneben Originalarbeiten aus Zeitschriften oder Tagungsbänden).

Integration und dauerhafte Verwaltung von inhaltlichen Komponenten obliegt einem LMS. Um eine Sammlung einzelner Einheiten überhaupt integrieren und einem Lernenden in einheitlicher Weise präsentieren zu können, sind naturgemäß Austauschformate erforderlich, die Inhalte vergleichbar strukturieren und darüber eine solche Zusammensetzung ermöglichen; besser noch werden bereits bei der Erstellung des Materials *Standards* eingehalten, die schon eine gewisse Akzeptanz besitzen. Die Entwicklung derartiger Standards ist daher ebenfalls als inhaltliche Anforderung zu sehen. Es wird niemanden wundern, dass an dieser Stelle wieder XML ins Spiel kommt, wovon weiter unten noch die Rede sein wird.

12.2.3 Organisatorische Anforderungen

Als Nächstes diskutieren wir *organisatorische Anforderungen* an ein E-Learning-System, wobei wir uns hier insbesondere auf die Sicht der Lernenden und damit (im Sinne von Abbildung 12-1) auf wünschenswerte Eigenschaften des Laufzeitsystems konzentrieren. Wir unterscheiden hier die *Vorbereitung*, die *Durchführung* und die *Nachbereitung* eines virtuellen Lehrbetriebs. Es sei bemerkt, dass wir uns hier auf das eigentliche Lernen sowie auf Lehrveranstaltungen konzentrieren und von Aspekten der Organisation eines Lehrbetriebs (wie z.B. Überwachung einer Prüfungsordnung oder Abrechnung von Kursgebühren) absehen.

Bei der *Vorbereitung* des Lernens muss es zunächst möglich sein, durch eine Einstufung oder einen Test entweder durch einen Lehrenden oder durch den Lernenden selbst zu klären, welche Vorkenntnisse im Hinblick auf die gewünschten Inhalte vorliegen oder welche Ziele der Lernende mit der Kursteilnahme verfolgt. Als Beispiel denke man wieder an die Traffic School, bei der die Teilnehmer sowohl erfahrene Verkehrsteilnehmer als auch Anfänger sein können. Dabei entsteht ein Lernprofil, das während des Lernens erweitert und vervollständigt wird; konzeptionell führt dies auf eine Benutzermodellierung, auf die wir in

Abschnitt 12.3 eingehen werden. Sodann müssen den Benutzern des Systems Möglichkeiten zur Kursauswahl bzw. zur Auswahl von Inhalten geboten werden. Im Idealfall erfolgt eine solche Auswahl interaktiv in der Form, dass das System eine vom Benutzer gewünschte Auswahl mit dessen Profil abgleicht und gegebenenfalls Vorschläge zur Modifikation oder Ergänzung der Auswahl macht. Es sollte in dieser Phase auch eine Definition von Lernzielen erfolgen, um später testen zu können, ob diese erreicht wurden. Sodann kann eine Konfiguration bzw. ein Buchen eines Kurses oder von Lernmodulen bzw. das Erstellen eines Lehrplans erfolgen; schließlich sollte der Lernende seine Lernumgebung in gewissem Umfang selbst einrichten können (was ist im Browser zu sehen, was wird nur auf Wunsch eingeblendet etc.), etwa im Stil des »konfigurierbaren Schreibtischs« des an der Universität Münster entwickelten Systems VISUM [visum].

Versteht man »Lernen« (im weiteren Sinn) auch als die kontinuierliche Information von Mitarbeitern etwa über ein Unternehmensportal, entfällt die gerade skizzierte Form der Vorbereitung; diese wird jedoch eventuell ersetzt durch Mitarbeiterprofile, in welchen ihre Lernleistungen protokolliert werden oder festgehalten ist, welche Fachkenntnisse der Einzelne einbringen kann.

Eine *Durchführung* des Lernens ist davon abhängig, ob das Lernen synchron oder asynchron erfolgt. Bei einem synchronen Lernen kann man von der Existenz eines realen oder virtuellen Klassenraums ausgehen, in welchem ein Tutor die Lernenden anleitet; in dieser Situation sind nicht alle der unten genannten Funktionalitäten erforderlich.

Ein asynchrones Lernen, bei welchem der Lernende auf sich gestellt ist, erfolgt im Rahmen der gerade erstellten Lernumgebung relativ zu einer aktuellen To-Do-Liste, welche in Abhängigkeit vom aktuellen Kontext nächste Schritte angibt. Wichtige Hilfsmittel, die einem Lernenden während des Lernvorgangs zur Verfügung stehen müssen, sind Kommunikationsmöglichkeiten und Möglichkeiten zur Recherche. So sollte es möglich sein, während einer Sitzung mit einem Tutor Kontakt aufzunehmen, der bei Fragen oder anderen Problemen unmittelbar hilft. Erfahrungen zeigen, dass es für die Lernenden demotivierend ist, wenn bei einem (vermeintlich) ernsthaften Problem keine Hilfe angefordert werden kann. Hilfreich sind ferner ein Chat-Room, in welchem Kontakt mit anderen Lernenden gehalten werden kann, E-Mail vom und zum Lernenden, die automatisierte Zustellung eines Newsletters sowie die Möglichkeit des Zugriffs auf ein Diskussionsforum oder auf eine FAQ-Liste. Möglichkeiten zur Recherche sollten einerseits eine spontane oder Katalog-basierte Suche im Web umfassen und andererseits die Suche anhand einer vorgegebenen Liste für den jeweiligen Inhalt wichtiger Links. Sinnvoll ist auch der Zugriff auf eine eigene oder fremde digitale Bibliothek, ein Glossar, ein Lexikon sowie auf Begleitdokumente wie Dokumentationen oder Handbücher, sofern diese relevant sind. Schließlich kann es dem Lernfortschritt zuträglich sein, wenn die Lernenden noch während des Lernens vom System Feedback über ihren Fortschritt erhalten. Spontane Tests können die Seriosität des Lernens verbessern; geplante Tests dienen der Messung des Lernfortschritts, aber auch der vergleichenden Einstufung in Bezug auf andere Bear-

beiter gleicher Inhalte. Man könnte an dieser Stelle sogar an eine Zeitmessung denken, welche bei jeder Lernsitzung aktiviert wird und z.B. den Zeitverbrauch für das Lesen einer Seite, den Zeitverbrauch pro Kurseinheit, oder den pro Test misst. Als letzte Komponente für eine Durchführung des Lernens kann in Abhängigkeit von den Inhalten eine Experimentierumgebung vorgesehen werden, in welcher Fallstudien, Simulationen oder Plan- sowie Rollenspiele vorgehalten werden.

Im Rahmen der *Nachbereitung* des Lernens sind Abschlusstests, elektronische Klausuren sowie Wiederholungsfragen denkbar. Fragen der Authentifizierung von Prüflingen bei Online-Prüfungen (etwa durch biometrische Verfahren) müssen dazu geklärt werden. Wie in realen Szenarien kann man ferner vorsehen, dass neben einem Ersttest bei Nichtbestehen ein Wiederholungstest absolviert werden kann. Am Ende sollten die Ausstellung eines Zertifikats durch die betreffende Organisation sowie eine Kursevaluation durch den Teilnehmer stehen.

Die gerade beschriebenen organisatorischen Anforderungen (im Sinne von Abbildung 12-1 an die *Laufzeitumgebung*) sind in Abbildung 12-2 zusammengefasst, wobei wir sie in die Kategorien *Kommunikation*, *Recherche*, *Experiment* und *Lernkontrolle* unterteilt haben. Klar sollte sein, dass die in dieser Abbildung gezeigte Umgebung je nach Anwendung eingeschränkt werden kann, etwa in dem betrieblichen Umfeld, in welchem Lernen kontinuierlich, in kleineren Einheiten als Kursen und im Allgemeinen nicht zu dedizierten Zeiten erfolgt. Außerdem zeigt die Abbildung nicht den »virtuellen Klassenraum«, in welchem sich ein kollaboratives Lernen in der Gruppe durchführen lässt, ohne dass sich die Gruppe am selben Ort befindet.

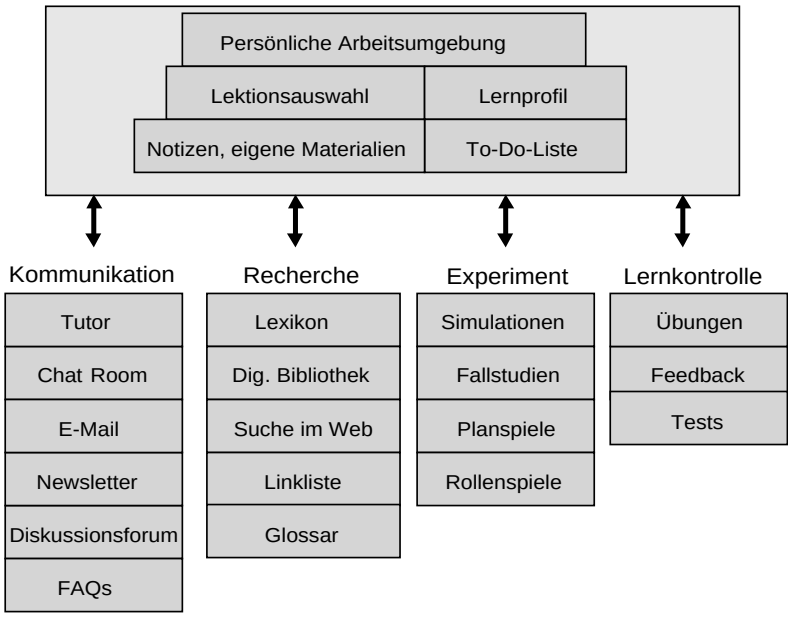


Abb. 12-2: *Organisation einer Laufzeitumgebung für Lernende.*

12.2.4 Technische Anforderungen

Ein Web-basiertes Lernsystem basiert heute im Allgemeinen auf einer Client-Server-Architektur und unterstützt serverseitig meist zahlreiche Anwendungen, darunter den Betrieb eines Web-Servers, die Anbindung von Fremdsystemen sowie den Betrieb eines Datenbank-Servers. Abbildung 12-3 zeigt diese allgemeine Architektur.

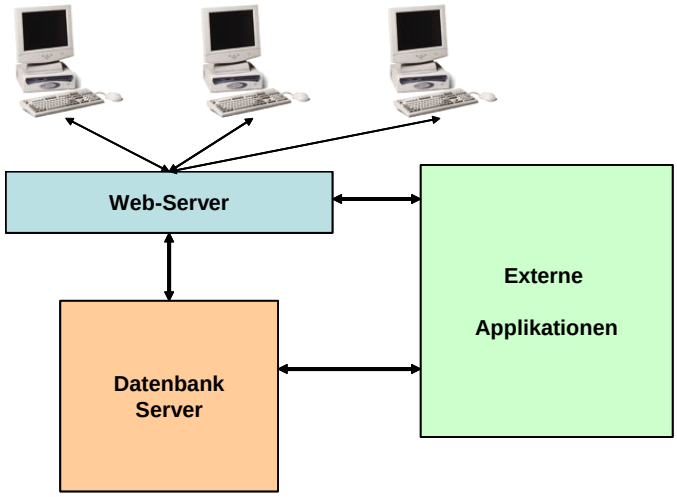


Abb. 12-3: *Grobarchitektur eines Lernsystems.*

Häufig werden Thin-Clients verwendet, so dass der Benutzer (etwa der Student in seiner häuslichen Umgebung) keine hohen technischen Anforderungen zu erfüllen hat, bevor er mit dem Lernsystem arbeiten kann. Man kann davon ausgehen, dass diese Anforderungen durch weiter sinkende Hardware-Preise und die fortschreitende Entwicklung von breitbandiger Verkabelung (bis hin zur Vision des allgegenwärtigen so genannten »Evernet«) an Bedeutung verlieren werden. In betrieblichen Umgebungen ist sie bereits heute nicht gravierend.

Andererseits sind serverseitig, wie in Abbildung 12-1 bereits gezeigt, Autorentätigkeiten von administrativen Arbeiten am System und von applikativen zur Laufzeit zu unterscheiden. Schließlich muss das System z.B. Sicherheitsanforderungen genügen. So unterliegen z.B. in einer Hochschule Prüfungsunterlagen bestimmten Handhabungsvorschriften; bei einer Web-basierten Klausur ist eine Authentifizierung der Klausurteilnehmer erforderlich. In einem Unternehmen ist die Speicherung personenbezogener Daten mit dem Betriebsrat abzustimmen. Ferner kann man sich unterschiedliche Distributionsformen vorstellen, die neben dem Internet oder einem Intranet auch traditionelle CD-ROMs oder sogar Druckmedien umfassen.

Wir werden uns im übernächsten Abschnitt mit der technischen Realisierung der Komponenten eines Lernsystems ausführlicher befassen und dabei insbeson-

dere den Zusammenhang zwischen den Abbildungen 12-1 und 12-3 genauer herausarbeiten.

12.3 Konzepte von E-Learning-Systemen

In diesem Abschnitt wollen wir uns mit *Konzepten* befassen, welche derzeit im Kontext von E-Learning und Web-basiertem Lernen diskutiert werden, wobei wir uns an der bisher bereits verwendeten Einteilung in Inhaltliches, Organisatorisches und Technisches orientieren. Bei den inhaltlichen Konzepten wirkt sich die Anforderung nach Modulen, die zu größeren Einheiten zusammensetzbar, aus unterschiedlichen Quellen integrierbar und wieder verwendbar sein sollen, zentral aus; man betrachtet hier so genannte *Lernobjekte* (*Learning Objects*), welche diese Eigenschaften besitzen sollen. Bei den organisatorischen Konzepten liegt eine Datenbankunterstützung nahe, über welche die hier relevanten Anforderungen bis hin zur Unterstützung von Workflows abgedeckt werden können. Zur Erfüllung der technischen Anforderungen werden unter anderem Web-Server, Dokumentenmanagementsysteme, aber auch Konzepte der Mensch-Maschine-Kommunikation wie z.B. Benutzermodellierung herangezogen.

12.3.1 Lernobjekte

Als Arbeitsdefinition wollen wir unter einem *Lernobjekt* eine unabhängige Einheit digitalisierten Lerninhalts verstehen, welche potenziell in verschiedenen Kontexten verwendet werden kann. Dieser erste Ansatz wird in der Literatur auf verschiedene Weisen konkretisiert, was hier kurz behandelt werden soll. Es sei allerdings schon vorab bemerkt, dass man im Zusammenhang mit E-Learning für ein Lernobjekt nicht den aus der Informatik bekannten Objektbegriff zugrunde legt, also das Verständnis einer Entität mit Struktur *und* Verhalten, Kapselung, eventuell Spezialisierung usw.; stattdessen versteht man hier darunter lediglich eine strukturelle Einheit, welche einen bestimmten Inhalt repräsentiert.

[Barr01] beschreibt die von Cisco Systems, Inc. für sämtliche internen und kundenbezogenen Aus- und Fortbildungsmaßnahmen verwendete Interpretation der *Reusable Learning Objects* (RLOs). Unter einem RLO wird dabei eine diskrete, wieder verwendbare Sammlung von Inhalten verstanden, welche der Darstellung und Vermittlung eines einzelnen Lernziels dient. Ein RLO hat einen fest vorgegebenen Aufbau bestehend aus *Overview* am Anfang, *Summary*, gefolgt von *Assessment* und dazwischen eine Sammlung von fünf bis neun *Reusable Information Objects* (RIOs); jedes RIO wiederum besteht aus einem inhaltlichen Teil, einem Übungsteil und einem Auswertungsteil. Im Hinblick auf klassische Ausbildung entspricht ein RLO etwa einer Unterrichtsstunde, ein RIO einem einzelnen Abschnitt darin. Wesentlich ist hier einerseits der teilweise feste, teilweise variable Aufbau von RLOs; darüber hinaus gibt es weiter gehende Vorgaben für den inhaltlichen Aufbau, welche dem Lernenden die Orientierung (an der gleich

bleibenden Struktur eines Lernobjekts) und dem Autor eines solchen Objekts die Erstellung erleichtern sollen. Ein *Overview* z.B. hat die folgenden sechs inhaltlichen Anteile:

1. Introduction
2. Importance
3. Objectives
4. Prerequisites
5. Scenario
6. Outline

Von diesen ist lediglich die Szenariobeschreibung optional. Analoge Schablonen existieren für RIOs, bei welchen die fünf Typen *Concept Fact*, *Procedure*, *Process* und *Principle* unterschieden werden, für deren bereits genannte Dreiteilung wiederum strukturelle Vorgaben greifen.

Vergleichbare Empfehlungen existieren auch von anderen Autoren bzw. in anderen Systemen, etwa [Down01]. Wiederverwendbarkeit von elektronisch vorliegenden Lernobjekten und Lernmodulen wird auch von [TrLi01] sowie von [ElFS01] diskutiert. Da es sich bei einer Vorstrukturierung de facto um die Festlegung von *Metadaten* handelt, ist es an dieser Stelle nicht verwunderlich, dass XML als Austauschformat zum Einsatz kommt; hierauf werden wir in Abschnitt 12.4 zurückkommen. Eine theoretische Untersuchung zu Lernobjekten und deren Sequenzierung liefert [Wile00].

12.3.2 Modellierung von Lernabläufen

Neben der Modellierung von Lernobjekten, etwa durch Vorgabe einer Strukturierung zur Unterstützung von Zusammensetzbarkeit sowie von Wiederverwendbarkeit, sind im Kontext von Lernsystemen weitere Aspekte zu modellieren. In diesem Abschnitt wollen wir kurz auf die Modellierung von *Lernabläufen* eingehen, wozu man Modellierungskonzepte verwenden kann, welche im Bereich der Prozessmodellierung eingesetzt werden.

Wünschenswert ist eine Integration von ablaufbezogenen Aspekten, die beispielsweise mittels Petri-Netzen oder verwandten Notationen dargestellt werden können, und dokumentenbezogenen Aspekten, die etwa mit XML beschrieben werden können. Die so genannten XML-Netze, eine spezielle Variante höherer Petri-Netze, sind zu diesem Zweck entwickelt worden (für eine ausführliche Beschreibung siehe [LeOb01]). Eine integrierte Beschreibung von Lernabläufen und Lernobjekten ermöglicht beispielsweise die simulative Überprüfung, ob ein vorgesehener individueller Lernablauf mit einer vorgegebenen Prüfungsordnung verträglich ist.

12.3.3 Datenbankunterstützung

In diesem Abschnitt behandeln wir die Komponenten, die man typischerweise in einer E-Learning-Umgebung vorfindet, aus einer organisatorischen Perspektive. Exemplarisch wollen wir dazu lediglich zwei Vorschläge vorstellen, welche sich ohne weiteres in die von uns eingangs vorgenommene Dreiteilung einordnen lassen; weitere finden sich z.B. in den Übersichten [wbt1, wbt2]. [GuOt97] beschreiben intelligente Lehr- und Lernsysteme, die aus unterschiedlichen wissensbasierten Komponenten bestehen:

1. Ein *Expertenmodul* besitzt explizites und implizites Wissen, um den jeweiligen Lerngegenstand in Form von Fakten und Regeln sowie über Problemlöseregeln beschreiben zu können. Es kann ferner für die Erzeugung und Korrektur von Aufgaben herangezogen werden.
2. Ein *Schülermodul* wertet mit Unterstützung des Experten individuelle Lernwege bzw. Lernabläufe aus. Es speichert in einem Schülermodell von Adressaten benutzte Fakten und Regeln sowie Erkenntnisse über dessen Fähigkeiten.
3. Ein *Tutormodul* besitzt Wissen über unterschiedliche Lehr- und Lernstrategien und beeinflusst den Ablauf eines Lernprozesses sowie das Lerntempo in Abhängigkeit vom Wissen über den Lernenden; es gibt ferner Feedback und Erklärungen.
4. Ein *Kommunikationsmodul* stellt eine geeignete Benutzerschnittstelle zur Verfügung.

[FeSc01] nennen die folgenden Komponenten der im Rahmen von [vhb] entwickelten Plattform I^2LU, man vergleiche auch [FHSU00]:

1. Die *Studierumgebung* bietet Wissensvermittlung und Übungsaufgaben an, und zwar in Form virtueller Selbstlernsituationen, virtueller Fachvorträge oder virtuellem Frontalunterricht; sie ist zum Selbstlernen und zum Lernen in Gruppen gedacht.
2. Die *Recherchierumgebung* bietet Wissensvermittlung im Stil einer Enzyklopädie und eignet sich damit für explorativen Wissenserwerb.
3. Die *Experimentierumgebung* stellt einen Experimentierbaukasten mit Simulationen, Planspielen, Fallstudien oder Rollenspielen zur Verfügung und dient dem Erwerb von Problemlösungsfähigkeit.
4. Die *Kommunikationsumgebung* ermöglicht den Teilnehmern den Zugriff auf Nachrichten- oder Konferenzsysteme oder Gruppeneditoren und unterstützt E-Mail, Videokonferenz, Chat, News und Whiteboard.
5. Die *Planungsumgebung* dient der Steuerung von Lernprozessen bzw. Lernabläufen durch Lehrende und Lernende über Werkzeuge wie Terminverwaltung, Lernwege- oder Projektplanung.
6. Die *Verwaltungsumgebung* erfasst die Teilnehmer mit ihren Profilen und dient der Zugangsverwaltung.

Bei einem *Web-basierten* System hat der Web-Server Anfragen von Clients zu beantworten, der Datenbank-Server muss dagegen unter Umständen mehrere der Funktionen abdecken, die wir bereits in den Abbildungen 12-1 und 12-2 angesprochen haben; dies ist in Abbildung 12-4 angedeutet.

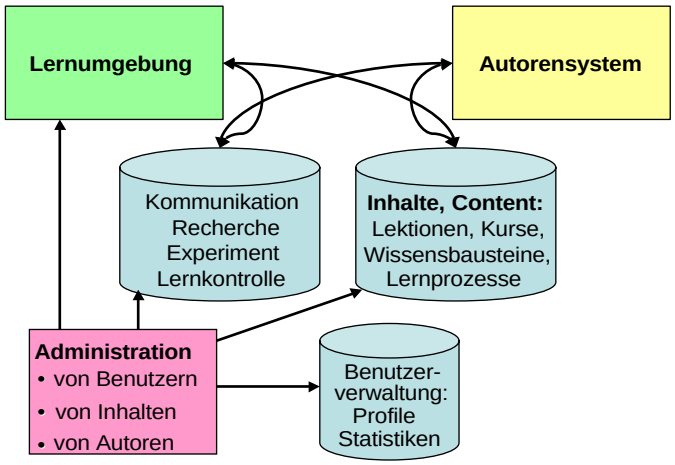


Abb. 12-4: *Datenbankorientierte Sicht auf ein Lernsystem.*

Abbildung 12-4 zeigt neben den bereits beschriebenen Systemkomponenten die verwendeten Datenbanken: Die wichtigste Datenbank enthält die Inhalte des Systems, also Lektionen, Kurse, Wissensbausteine und ist naturgemäß zielgruppengerichtet und auf bestimmte Fächer konzentriert. Wichtig erscheint uns, dass im Idealfall in dem System nicht nur Inhaltsbausteine hinterlegt sind, sondern auch prozesstechnisch abgebildet ist, auf welche Weise und in welchen Reihenfolgen diese zu bearbeiten sind, so dass im Inhaltsteil des Systems auch Lernprozesse abgebildet sind.

Bei den in einem E-Learning-System verwendeten Datenbanken kann es sich um *zentralisierte* Datenbanken handeln, welche räumlich an einer Stelle untergebracht sind (um z.B. zentral administriert werden zu können) und auf die über das Internet zugegriffen werden kann. Die Daten können auch verteilt gehalten werden, z.B. auf unterschiedlichen Servern im Web, so dass es sich bei einem Kurs dann lediglich um eine virtuelle Integration von Modulen handeln kann. Unter der Vision des »Evernet«, also des allgegenwärtigen Internet, in welches man permanent und ortsunabhängig eingeloggt ist, wird man ferner neue Formen des mobilen Lernens unterstützen müssen. Die Datenbanken eines Lernsystems müssen dann in der Lage sein, dem Benutzer zu folgen und Inhalte auf einem zu dessen aktuellem Aufenthaltsort »nahen« Server vorzuhalten.

Weitere Datenbank-Aspekte lassen sich anhand der von uns oben bereits verwendeten Dreiteilung angeben: Hinsichtlich der *Lernersicht* auf ein E-Learning-

System ist die Unterstützung multimedialer Inhalte wesentlich (HTML-Seiten, aber auch Graphiken, Diagramme, Bilder, eventuell Bewegtbilder, Dia- und Slide-Shows, Aufzeichnungen von Vorträgen). Hier bieten moderne Datenbanksysteme durchweg Erweiterungen an (in Form so genannter *Extender*, *Cartridges* oder *Data Blades* usw.); diese Erweiterungen bieten über den Standardumfang von SQL hinaus neue Datentypen (wie `audio`, `video`, `image`, `text`, `xml` usw.) sowie Familien von Funktionen zu deren Verarbeitung an, die wie systemseitig vordefinierte Datentypen verwendbar sind; man vergleiche z.B. [Cham99], [SiKS02] oder [Voss00].

Aus der *Autorensicht* sind insbesondere Werkzeuge für die Erstellung von Inhalten wesentlich, wobei auch hier Datenbanken, die multimediale Datentypen unterstützen, von zentraler Bedeutung sind. Darüber hinaus benötigt ein Autor einer Lerneinheit, die über ein elektronisches Lernsystem angeboten werden soll, mächtige Editierwerkzeuge und eventuell Werkzeuge zur Erstellung von Videos oder Filmen. Schließlich sind aus der *Administratorsicht* vor allem die Laufzeitumgebung sowie das Verwaltungssystem relevant. Hier sind insbesondere Sicherheitsanforderungen zu erfüllen, die es z.B. ermöglichen, Prüfungsergebnisse fälschungssicher und langfristig reproduzierbar zu speichern, was z.B. über geeignetes Session-Management oder verschlüsselte Verbindungen erreicht werden kann.

12.3.4 Benutzeradaption am Beispiel AHA

Ein wesentlicher Aspekt eines elektronischen oder Web-basierten Lernsystems ist dessen Fähigkeit, sich an unterschiedliche Benutzer, ihre Vorkenntnisse, Lernstile, Lerntempi und Fortschritte individuell anzupassen. So können z.B. unterschiedliche Benutzer die mit einer Lerneinheit verbundenen Tests zu unterschiedlichen Zeiten absolvieren oder Lerninhalte in unterschiedlicher Ausführlichkeit präsentiert bekommen.

Exemplarisch gehen wir hier auf das AHA-System der TU Eindhoven als Beispiel eines in diesem Sinne *adaptiven* Systems ein, vgl. [DeCa98a, b]. *AHA* steht für »Adaptive Hypermedia Applications«. In diesem System macht man sich grundsätzlich die Technik der Hypertexte zur Aufbereitung von Lerninhalten zunutze, so dass ein Leser bzw. Benutzer z.B. vom Inhaltsverzeichnis eines Skripts mit einem Klick in ein gewünschtes Kapitel springen kann oder vom Index am Ende eines Kurses direkt an die Stelle, an welcher ein bestimmter Begriff erläutert wird. Wesentlich ist jedoch, dass neben der Navigation eine Adaption (sowohl von Links als auch von Inhalten) stattfindet. Falls ein Kurs z.B. vorsieht, dass jeder Teilnehmer zunächst das erste Kapitel durcharbeiten muss, etwa weil dieses Installationshinweise für eine bestimmte, für den Kurs benötigte Software enthält, so wird beim ersten Start des Systems nur dieses Kapitel frei geschaltet; alle anderen Links des betreffenden Kurses sind deaktiviert. Erst wenn das System merkt, dass der Benutzer in dieses Kapitel mindestens einmal verzweigt hat, werden beim nächsten Besuch der Startseite weitere Kursteile frei geschaltet bzw. die Links zu

diesen aktiviert. Durch diese Anpassung (in diesem Beispiel an den Lernfortschritt des Benutzers) wird gesteuert, auf welche Teile eines Kurses aktuell Zugriff gewährt wird oder was ein Benutzer als Nächstes bearbeiten soll. Ein anderes Beispiel für die Verwendung von Adaption in AHA ist das Anbringen von kurzen, weiterführenden Erläuterungen innerhalb eines durchzuarbeitenden Textes: Wird auf einen mit einem Link hinterlegten Schlüsselbegriff, zu dem eine Erläuterung vorhanden ist, geklickt, so wird zu dieser verzweigt; nach Rückkehr in den Haupttext ist der Hinweis auf die Erläuterung jedoch ganz oder in Teilen verschwunden. Das System merkt sich die Tatsache, dass der Benutzer die Erläuterung gesehen hat, und blendet sie anschließend aus.

Die Adaption selbst basiert auf einer *Benutzermodellierung*, bei welcher explizit durch den Benutzer selbst oder automatisch durch das System beschrieben (und dynamisch während des Lernens aktualisiert) werden kann, welche Fertigkeiten ein Benutzer bereits hat, welche Präferenzen bestehen oder welche Erscheinungsform das System ihm oder ihr gegenüber haben soll. Systemseitig wird ein Benutzer als Objekt aufgefasst, welches durch eine Reihe von Attributen beschrieben wird, denen Werte zugeordnet sind; eine Adaption verändert diese Werte. In der ursprünglichen Version von AHA besteht ein solches Objekt aus einer Menge *boolescher* Variablen, welche alle mit dem Wert »falsch« initialisiert werden. Nach dem Durchlesen einer Seite oder dem Bestehen eines Tests können sich die Werte einer oder mehrerer Variablen in »wahr« ändern. Die Verwendung boolescher Variablen impliziert, dass die Benutzermodellierung feingranular erfolgt, da man z.B. Benutzerklassifikationen wie »Anfänger«, »Zwischenstadium« oder »Experte« jeweils durch eine eigene Variable modellieren muss, wobei nicht alle Wertekombinationen sinnvoll sind (in diesem Beispiel etwa die Kombination »1 0 1«, die besagen würde, dass ein Benutzer sowohl Anfänger als auch Experte ist). Alternativ kann eine Benutzermodellierung grobgranular erfolgen und dabei mit weniger Attributen auskommen, welche dann allerdings jeweils mehr als zwei Werte annehmen können. Die aktuelle Wertebelegung der booleschen Variablen wird über Abhängigkeiten zwischen diesen sowie zwischen den Themenschwerpunkten eines Hyperdokuments gesteuert. Diese Themen sind ebenfalls an boolesche Variablen gebunden, z.B. »readme«, »introduction«, »history« oder »definitions«. Der Autor eines Dokuments muss festlegen, welche Seite vor oder nach welcher anderen gelesen werden muss. Dazu enthält jede Seite in ihrem Header eine Liste von Variablen, die gesetzt sein müssen; enthält ein solcher Header z.B. die Angabe »readme = 1«, so muss die mit der Variablen »readme« assoziierte Seite (die technische Instruktionen enthält) gelesen worden sein, bevor die aktuelle Seite (z.B. die Startseite eines Kurses) aufgerufen werden kann. Ein Lesen der als Voraussetzungen geforderten Seite wiederum bewirkt Veränderungen an den Benutzervariablen (z.B. wird »Instruktionsseite gelesen« gesetzt).

Die einzelnen Einträge in ein Benutzermodell werden relativ zu einem *Diskursbereichsmodell* vorgenommen, welches beschreibt, wie der Informationsgehalt der betreffenden Anwendung strukturiert ist, welche Beziehungen zwischen den

einzelnen Bereichen bestehen, aus denen eine Anwendung (ein Kurs) zusammengesetzt ist und wie die vorkommenden Konzepte an Informationsfragmente, Dokumente und Seiten gebunden sind. Grundsätzlich ist es bei AHA dem Autor eines Dokuments überlassen, die Themen des Dokuments zu identifizieren und die Abhängigkeiten zwischen diesen festzulegen. Dabei sind jedoch Vereinfachungen möglich, etwa durch eine Verwendung konditionaler Links, die im Dokument nicht ständig zu sehen sind; ferner ist bei vielen Dokumenten wesentlich, dass die Startseite am Anfang und die Seite mit dem Abschlusstest am Ende gelesen wird, nicht aber die genaue Reihenfolge, in der die dazwischen liegenden Textteile bearbeitet werden.

In einer neueren AHA-Version wird der Benutzer durch eine relationale Tabelle beschrieben, deren Attribute auch nicht boolesche Werte annehmen können. Beispiele sind die Attribute *knowledge*, *read* und *ready (to be read),* von denen Ersteres die Werte *learnt, well-learnt, not known* usw. annehmen kann, vgl. [DeBr99] sowie [WuHD99]. Zahlreiche andere Attribute sind denkbar, etwa *knowledge decay* (das angibt, wie schnell der Benutzer das betreffende Konzept vergessen hat) oder *expiration time* (die angibt, wann der Inhalt einer Seite als veraltet angesehen werden soll). Die *read*-Attribute sind dabei den einzelnen HTML-Seiten zugeordnet, aus denen ein Dokument oder ein Kurs besteht. Für jede solche Seite hat *read* den Wert »wahr« oder »falsch« in Abhängigkeit davon, ob der betreffende Benutzer die Seite bereits gelesen hat oder nicht (wobei bereits ein Zugriff auf eine Seite als Lesen dieser interpretiert wird). Die entsprechende Adaptionsregel lautet:

$$access(P) \rightarrow P.read := true$$

Die *knowledge*-Attribute existieren für die einzelnen Konzepte des Diskursbereichsmodells, welches aus *Konzepten* (im Wesentlichen die HTML-Seiten eines Dokuments), *Fragmenten* und *Seiten* besteht und die Beziehungen zwischen diesen Bestandteilen beschreibt. So stellt z.B. »*generates*« eine Beziehung zwischen einer Seite *S* und einem abstrakten Konzept *K* dar in dem Sinne, dass ein Lesen von *S* Wissen über *K* erzeugt. Wesentlich ist, dass die vom AHA-System geleistete Adaption durch die relationale Modellierung auf Trigger im unterliegenden Datenbanksystem abgestützt werden kann.

Es sei bemerkt, dass zu AHA vergleichbare Systeme auch an anderen Stellen entwickelt werden; man vergleiche z.B. [Hara99] oder [ShSC01].

12.3.5 Wissensmanagement in KBS

Ein anderes System zur Modellierung, Organisation und Wartung verteilter Hypermedia-Ressourcen zum Zwecke der Unterstützung von Web-basiertem Lernen ist das an der Universität Hannover entwickelte *KBS Adaptive Hyperbook*. Ein *Hyperbook* ist dabei zunächst eine Sammlung von Hypertext-Dokumenten, welche hier jedoch zu einem Informations-Repository erweitert ist, das unter Rückgriff auf semantische Modelle und Metadaten Lernmaterialien integriert und

(durch Adaption) personalisiert. Als Domain-Modell wird ein Metamodell verwendet, welches die zentralen Klassen »Concept« und »Relation« enthält. Ein Autor eines Kurses muss ein solches Domain-Modell konstruieren und die darin enthaltenen Klassen anschließend mit Lehrmaterial instanziieren. Daneben enthält KBS ein Studierendenmodell, über welches zur Laufzeit die eigentliche Adaption gesteuert wird; so wird z.B. in Abhängigkeit vom gewählten Lernziel und Lernprojekt eine Bearbeitungssequenz vorgeschlagen. KBS enthält ferner eine Bibliothek von Projekten, welche mit den »Knowledge Items« indexiert sind, die man für eine erfolgreiche Bearbeitung des Projekts verstanden haben muss. Ein erstes konkretes Hyperbook wurde für einen Programmierkurs entwickelt. Das System hat ursprünglich O-Telos als Wissensrepräsentationssprache benutzt und diese inzwischen durch XML Schema ersetzt; man vergleiche [QuNe00] sowie [HeNe99].

12.4 Realisierungen

In diesem Abschnitt gehen wir auf exemplarische Realisierungen von Web-basierten Lernsystemen ein, wobei wir keinen Anspruch auf Vollständigkeit erheben und grundsätzlich wieder unserer Dreiteilung in Autorensystem, LMS und Laufzeitsystem folgen wollen. Wir behandeln zunächst kommerzielle Systeme überwiegend in Form von Hinweisen und gehen lediglich auf verschiedene nichtkommerzielle Entwicklungen näher ein.

12.4.1 Autorensysteme

Hinsichtlich der Erstellung von Lernobjekten und Lernabläufen wird heute von vielen Systemen Unterstützung geboten; verschiedene Übersichten finden sich im Web z.B. unter [mar], [wbt1], [wbt2] oder [c2t2]. Gelegentlich werden von derartigen Systemen auch nur Teilaspekte der Kursentwicklung abgedeckt, insbesondere die Erstellung von Tests; man vergleiche hierzu z.B. [ets] oder [hot]. Für Interessenten bieten viele Hersteller Testversionen an oder Demos im Web, etwa das Java-Applet *CourseDesigner* des Fraunhofer-Instituts Software- und Systemtechnik [isst] oder die funktionsreduzierte Version des ReadyGo *Web Course Builder* [ready]. Weitere exemplarisch zu nennende Systeme sind Macromedia Authorware, click2learn ToolBook, Lectora Publisher oder WebCT Vista.

12.4.2 Standardisierungsaktivitäten

Wie erwähnt wird die Erstellung von Lerninhalten und Lernobjekten allgemein als von zentraler Bedeutung für den erfolgreichen Einsatz eines Lernsystems angesehen, gilt jedoch gleichzeitig als zeitaufwändig und teuer, so dass Wiederverwendbarkeit und Zusammensetzbarkeit von Lernobjekten aus unterschiedlichen Quellen von hohem Interesse ist. Diesem tragen unterschiedliche Standardisierungs-

aktivitäten Rechnung, von denen einige wichtige hier genannt seien (siehe auch [Fisc01]):

- ADL, *Advanced Distributed Learning*, ist eine Initiative des amerikanischen *Department of Defense* sowie akademischer und industrieller Partner zur Entwicklung portabler Kurse; Hauptgegenstand ist die Entwicklung des SCORM (*Sharable Content Object Reference Model*) genannten Rahmenwerks, auf das wir unten noch eingehen; man vergleiche auch [adl].
- AICC, das *Aviation Industry Computer-Based Training Committee*, ist eine internationale Vereinigung professioneller Trainer, die gemeinsam Ausbildungsrichtlinien für die Luftfahrtindustrie erarbeiten [aicc]. Das AICC entwickelt Standards zur Erzielung von Interoperabilität von rechnergestützten Ausbildungsprodukten unterschiedlicher Hersteller.
- DCMI, die *Dublin Core Meta-data Initiative*, ist ein Forum zur Entwicklung von Metadaten-Standards für unterschiedliche Zwecke und Geschäftsmodelle sowie zur Festlegung spezifischer Metadaten-Vokabulare zur Ressourcenbeschreibung, vgl. [dcmi].
- Das IEEE *Learning Technology Standards Committee* [ltsc] erarbeitet technische Standards, Empfehlungen und Richtlinien für Software-Komponenten, Werkzeuge, Technologien und Entwurfsmethoden (siehe unten). Besonders zu erwähnen ist hier der Standard für *Learning Object Metadata* (LOM), welcher bereits auf große Akzeptanz stößt; man vergleiche [IEEE02].
- Das IMS (Instructional Management System) Global Learning Consortium definiert offene Architekturspezifikationen für Produkte aus dem Bereich des E-Learning, wie den *Learning Resource Meta-Data Best Practice and Implementation Guide* oder den *Content Packaging Best Practice Guide*, siehe [ims].

Dem SCORM-Vorschlag wird derzeit eine gewisse Favoritenrolle unter den Standardisierungsaktivitäten zugebilligt, denn SCORM beinhaltet ein Referenzmodell für Interoperabilität und Wiederverwendbarkeit von Inhalten, Technologien und Systemen und folgt der aus Abb. 12-1 bekannten Unterteilung. Die Implementierung eines Lernobjekts wird im Rahmen von SCORM als *Sharable Content Object* (SCO) bezeichnet; ein solches kann einen oder mehrere elektronische Inhalte umfassen und ist von einem LMS identifizierbar. SCOs werden durch Metadaten beschrieben, wobei SCORM den von IEEE LTSC stammenden LOM-Standard (*Learning Object Metadata*) verwendet. Wesentlich ist hier eine technische Sicht, die z.B. festlegt, welche Funktionsaufrufe ein SCORM-konformes LMS-API unterstützen muss. Die genannten Ziele sollen über XML-Schemata zum »Packen« von Lernobjekten erreicht werden, welche im Web zum Download bereit stehen. Diese definieren z.B. das Element <lom> mit den Subelementen <general>, <lifecycle>, <metametadata>, <technical>, <educational>, <rights>, <relation>, <annotation> und <classification>; Einzelheiten entnehme man [ADL01].

Das SCORM-Modell wird bereits von ersten Implementierungen unterstützt; so lassen sich etwa mit dem im Web frei verfügbaren LRN 3.0 Toolkit von Microsoft, vgl. [lrn], einfache Integrationen von Lernmaterialien vornehmen, deren durch das Werkzeug vorgenommene »Verpackung« mittels der von SCORM festgelegten Metadaten beschrieben wird. Ein SCORM-kompatibles Forschungsprojekt ist beispielsweise das spanische SimulNet, vgl. [AnRi01]. [Leid01] beschreibt eine Ontologie für pädagogisches Wissen, welche ebenfalls in XML und in SCORM darstellbar ist.

Es verwundert grundsätzlich nicht, dass sich auch in diesem Anwendungsbereich des elektronischen Lernens XML als Metadatenformat und Austauschplattform durchsetzt. Daher erwähnen wir als eine weitere Aktivität in diesem Zusammenhang die an der Open University of the Netherlands in Entwicklung befindliche *Educational Modeling Language* (EML), vgl. [Koop01]. In diesem XML-Derivat werden ähnlich wie bei den Cisco RLOs strukturelle Komponenten festgelegt, welche in einem hier »Unit-of-Study« genannten Lernobjekt enthalten sein sollen bzw. müssen. Eine vergleichbare Aktivität ist die an der Universität Passau entwickelte *Learning Material Markup Language* (LMML), vergleiche [SüFr01].

12.4.3 Nichtkommerzielle Entwicklungen

Seit etwa Mitte der 90er Jahre gibt es eine Vielzahl von universitären Aktivitäten in Richtung E-Learning [Simo01], von denen manche lediglich zum Ziel haben, klassische Unterrichts- sowie Lehr- und Lernformen multimedial zu gestalten, andere dagegen diese Formen durch neue Medien vollständig ersetzen wollen. Dies reicht von multimedial aufbereiteten Vorlesungen über Aus- sowie Weiterbildungskurse im Web bis hin zu »virtuellen Universitäten« wie der examenstechnisch an der Universität Frankfurt/Oder angesiedelten *Virtual Global University* [vgu], die ganze Curricula ausnahmslos in elektronischer Form anbieten. In diesem Abschnitt beschreiben wir einige Projekte aus dem universitären Umfeld, wobei wir uns auf Projekte aus dem Bereich der Informatik konzentrieren.

Im Projekt *VIROR* (Virtuelle Hochschule Oberrhein) betreiben die badischen Universitäten Freiburg, Heidelberg, Karlsruhe und Mannheim seit Mitte 1998 ein Pilotprogramm, welches den Aufbau eines multimedialen, hochschulübergreifenden Studienprogramms zum Ziel hat. Dabei wird insbesondere untersucht, in welchen Lernsituationen der Einsatz welcher multimedialer Mittel sinnvoll ist, welche Methoden in einer multimedialen Lehre sowie im Teleteaching sinnvollerweise zum Einsatz kommen und wie sich ein Standort übergreifendes Studienprogramm realisieren lässt. In Teilprojekten werden dabei die Aspekte Inhaltserstellung, Technik, Begleitung und Organisation bearbeitet; ferner werden unterschiedliche Lernszenarien untersucht, etwa die Übernahme kompletter Vorlesungen von einem Ort an einen anderen, multimediale Bausteine zur mehrfachen Verwendung zwecks Reduktion von Überschneidungen zwischen verschiedenen Veranstaltungen, interaktives Lehrmaterial und gemeinsame Gruppensemi-

nare. Bei synchronen Veranstaltungen, welche per Internet zeitgleich an verschiedene Orte übertragen werden, werden Audio- und Videodatenströme des Dozenten per *Authoring on the Fly* (AoF) aufgezeichnet, um dann für asynchrones Lernen wieder zur Verfügung gestellt werden zu können, vgl. [Schr00]. AoF bezeichnet dabei eine Umgebung, in welcher die aufgezeichneten Daten in direkt zugreifbare Objektlisten umgewandelt werden; das darin enthaltene Material kann dann selektiv abgespielt, aber auch ergänzt, analysiert oder mit anderem Material vernetzt werden. Einzelheiten entnehme man [HSKV01], [Kand99], [MüOt99] oder [Schr00].

Im Projekt *ViKar* (Virtueller Hochschulverbund Karlsruhe) entwickeln die sechs in Karlsruhe ansässigen Hochschulen einen gemeinsamen, virtuellen Campus mit dem Ziel, die Qualität des Präsenzstudiums durch elektronische, multimediale Mehrwertdienste zu erhöhen. Der virtuelle Campus ist in die neun Bereiche Anmeldung, Campusverwaltung, elektronische Bibliothek, Studienarbeitsplatz, Multimedia-Produktion, Kommunikation, Infokiosk, Lernserver und virtuelles Labor unterteilt. Für einen Benutzer sind die sechs verschiedenen Rollen *Studierender, Dozent, Autor, Studienadministrator, Kursleiter* oder *Verwalter* vorgesehen. Neben dieser elektronischen Nachbildung der Hochschulrealität werden auch Komponenten entwickelt, welche im realen Studium nicht denkbar sind, etwa ein persönlicher Studienassistent, der Hilfestellung geben kann bei Recherchen in Bibliotheken, Vermittlung von Experten, Erledigung von Verwaltungsaufgaben, Kommunikation mit anderen Personen auf dem virtuellen Campus oder Einholen neuer Informationen zu Veranstaltungen oder Prüfungen. Einzelheiten zu ViKar findet man z.B. bei [Bart01].

Es sei bemerkt, dass VIROR und ViKar Teile der *Virtuellen Hochschule Baden-Württemberg* sind [vhbw] und es eine Reihe weiterer Projekte mit vergleichbaren Zielsetzungen gibt, etwa den Virtuellen Campus Rheinland-Pfalz [vcrp] oder die Virtuelle Hochschule Bayern [vhb]. Das von [Ehre01] beschriebene Projekt *WINFOLine* (»Wirtschaftsinformatik Online«) hat die Bereitstellung eines universitätsübergreifenden Internet-Angebots auf dem Gebiet der Wirtschaftsinformatik zum Ziel. Virtuelle Lernwelten, die aus modularen Wissensbausteinen sowie Betreuungs- und Serviceangeboten bestehen, ersetzen dabei klassische Vorlesungen sowie Seminare. Zu Erfahrungen mit diesem Projekt vergleiche man [Bein01]. Die Virtuelle Hochschule Bayern [vhb] ist ein Verbundinstitut aller staatlichen Universitäten und Fachhochschulen in Bayern. Sie bündelt in ihrem Internetportal Lehrangebote z.B. in Informatik oder Ingenieurwissenschaften; man vergleiche hierzu [FeSc01].

Ein Verbundprojekt, das sich inhaltlich zunächst der wirtschaftswissenschaftlichen Lehre gewidmet hat, ist der Universitätsverbund Multimedia des Landes Nordrhein-Westfalen und sein Projekt *OpenUSS*, vgl. [Dobe00] oder [Grob01]. Im Rahmen dieses Verbundes wurde Software entwickelt, die inzwischen als Open Source unter [CaS] verfügbar gemacht wurde. [Grob00] beschreibt den Münsteraner Beitrag cHL (»Computergestützte Hochschullehre«) zu diesem Projekt, welcher insbesondere Komponenten, Werkzeuge und Anwendungen von Multi-

media in einem Referenzmodell darstellt und in verschiedene Teilbereiche (CAL, CAT, IAL, IAT) gegliedert ist. Wesentlich sind ein Granulatkonzept als Bezugsrahmen für die Entwicklung von Lerneinheiten (mit Bestandteilen wie »Intro«, »Text Study«, »Learning by Doing« oder »Check up«) sowie ein Administrationssystem mit E-Mail-Verteiler, Diskussionsforen und Archiv. [CoGr01] beschreiben darüber hinaus ein Konzept zur Konstruktion und Nutzung von multimedialen Modellen in der universitären, aber auch der betrieblichen Weiterbildung.

Im Paderborner Projekt *DISCO* (»Digitale Infrastruktur für Computerunterstütztes kooperatives Lernen«) geht es um eine Verknüpfung von Lern- und Arbeitsorten über eine geeignete Netzinfrastruktur sowie durch Web-basierte Serverstrukturen und eine geeignete Ausstattung von Lehrräumen; Lehrende und Lernende sollen in die Lage versetzt werden, jeweils benötigte multimediale Materialien an allen Lernorten abrufen und bearbeiten zu können. Besonderer Wert wird dabei auf die Ausgestaltung der »Lernorte«, also z.B. von Hörsälen gelegt, die idealerweise interaktiv sind und von jedem Arbeitsplatz einen Zugriff auf das Vorlesungsmaterial wie z.B. Folien gestatten, vgl. [Hamp01].

Am Lehrstuhl für Softwaretechnik der Universität Bochum wurde und wird das System *w3l* entwickelt [w3l]. Bei diesem System werden unterschiedliche Kurstypen unterschieden und verschiedene Lernstile unterstützt. Elementare Lerneinheit ist der so genannte Wissensbaustein, welcher mit einem Schwierigkeitsgrad versehen und im Zeitungsstil aufbereitet ist; nach einer Überschrift folgt der Inhalt, in welchem Attribute hervorgehoben sind, von denen aus in ein Glossar verzweigt werden kann. Ferner ist der Text mit Randmarken (»Definition«, »Beispiel«, »Notiz«, weitere) versehen, und es können Zusatzinformationen (wie Lernziele, Voraussetzungen, Version, Verweise) abgerufen werden. Zusammen mit eigenen Notizen eines Lernenden lassen sich Wissensbausteine zu einem individualisierten Textbuch zusammenstellen, das bei Bedarf als PDF-Datei gedruckt werden kann. Das System unterscheidet eine Lernenden- und eine Autorensicht; der Lernende erhält einen Browserzugang, wohingegen ein Autor spezielle Software installieren muss. Technisch basiert w3l auf dem Janus-OOA-Generator; das Klassenmodell enthält rund 50 Klassen, darunter »Wissensprofil«, »Test«, »Administrator«, »Glossar« oder »Literatureintrag«.

12.4.4 Die XLX-Plattform der Universität Münster

An der Universität Münster wird in der Arbeitsgruppe eines der Autoren die XLX-Plattform (»*eXtreme e-Learning eXperience*«) entwickelt. Diese dient bisher speziell der Unterstützung des *Übungsbetriebs* für Studierende der Wirtschaftsinformatik nach dem Vordiplom. Sie wird seit dem Sommer 2001 zur Durchführung des Übungsbetriebes technischer Vorlesungen (z.B. über Datenbanksysteme, Rechnernetze, XML) für Hörer nach dem Vordiplom eingesetzt. Bei XLX wird davon ausgegangen, dass Studierende an Präsenzvorlesungen teilnehmen, zu de-

nen Übungen angeboten werden. Diese beinhalten eine Vielzahl von Aktionen und Tätigkeiten, vom rein »rechnerischen« Lösen von Aufgaben etwa im Zusammenhang mit der Relationenalgebra, vgl. [Voss00], über die Formulierung von Anfragen und Programmen z.B. in SQL bis hin zur praktischen Arbeit mit Systemen. Die XLX-Plattform realisiert den Übungsbetrieb in Form einer geschlossenen Benutzergruppe, in welcher jedes Mitglied nach einer Registrierung ein eigenes Login mit Passwort erhält. Durch ein Login erhält der Teilnehmer Zugriff auf ein geschütztes und kontrolliertes Portal, welches Material zur jeweiligen Vorlesung (z.B. Links, Folien, Skripte), einen E-Mail-Verteiler, ein Diskussionsforum und vor allem den personalisierten Übungsbereich umfasst. Dieser Übungsbereich ist unterteilt in zwei Teilbereiche:

1. *Unbewerteter Testbereich*: Hier können Studierende ihre Fertigkeiten im Umgang mit vorlesungsrelevanten Techniken festigen (z.B. Formulieren von Anfragen in der Datenbanksprache SQL, vgl. [Voss00]; Transformation von XML-Dokumenten mit XSLT, vgl. [Ecks99] sowie [Kay00]), und sie können das Verständnis von in der Vorlesung vorgestellten algorithmischen Techniken vertiefen (Beispiele: Algorithmen der Datenbanksystemtechnik wie Optimierung von Anfragen mit Techniken der Relationenalgebra, Zwei-Phasen-Sperrprotokoll zur Synchronisation von Transaktionen, vgl. [WeVo02]).
2. *Bewerteter Übungsaufgabenbereich*: Hier finden die Studierenden Übungsaufgaben vor, welche vorlesungsbegleitend und zu bestimmten Terminen zu bearbeiten sind. Neue Aufgaben werden in bestimmten Zeitabständen und in inhaltlich sinnvoller Reihenfolge im Verlauf der Vorlesung zur Bearbeitung frei geschaltet. Nach Ablauf einer Bearbeitungsfrist werden keine Lösungen mehr angenommen. Zur Vorbereitung auf das eigentliche Lösen von Aufgaben dient der oben beschriebene Testbereich, in den jederzeit zurück gewechselt werden kann. Werden Aufgaben bearbeitet, so sind diese abzuschicken und können danach nicht mehr verändert werden. Die Aufgaben werden soweit wie möglich automatisiert bewertet.

Beim XLX-System [xlx] können sich wie bei zahlreichen anderen Systemen auch Benutzer (und Gäste) im Web registrieren und anmelden. Das System erkennt den Benutzer, er oder sie wird mit Namen begrüßt. Sodann ist der aktuelle Stand der kursbezogenen News zu sehen. Nach der Anmeldung lässt sich z.B. das bisherige Ergebnis im laufenden Übungsbetrieb einsehen (abgegebene Lösungen und deren Bewertung). Im Übungsbetrieb selbst bearbeiten die Studierenden zugewiesene, obligatorische Übungsaufgaben, daneben gibt es Zusatzaufgaben, die interessierten Studierenden die Gelegenheit geben, sich mit weiterführenden Fragestellungen zu beschäftigen. Schließlich stehen zwei »Spielwiesen« zur Verfügung, die transparenten Zugriff auf eine relationale Datenbank und einen XSLT-Prozessor ermöglichen.

Abbildung 12-5 zeigt einen als Petri-Netz modellierten Workflow für die studentische Bearbeitung einer Aufgabe im XLX-System: Nach der Anmeldung am System kann man eine zur Bearbeitung freigegebene Aufgabe auswählen und zu

dieser entweder eine Lösung testen oder eine solche zur Bewertung abgeben (»submit«). Testergebnisse sind archivierbar, während abgegebene Lösungen einer Bewertung zugeführt werden (die von einem Tutor vorgenommen wird, so dass der untere Teil des Workflows von einer anderen Person ausgeführt wird als der obere).

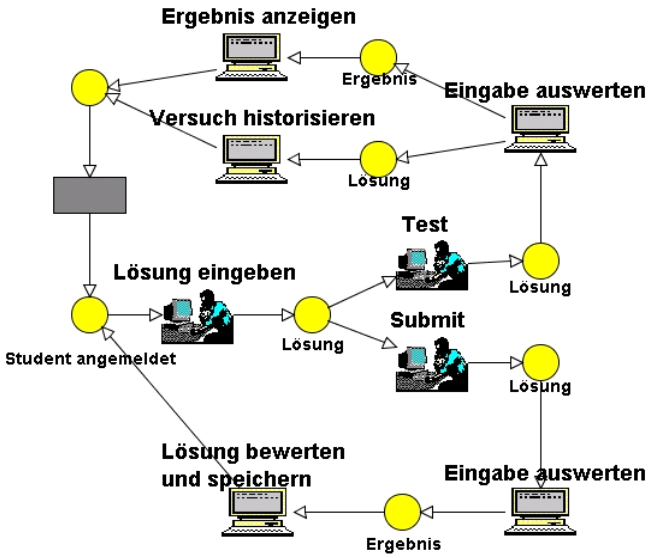


Abb. 12-5: *Aufgabenbearbeitungsworkflow im XLX-System.*

Bisher lassen sich Freitext-Aufgaben, Multiple-Choice-Aufgaben, SQL-Aufgaben und XSLT-Aufgaben im XLX-System erstellen und bearbeiten. Bei den ersten beiden Typen handelt es sich um Standardtypen, wie sie auch in anderen Lernsystemen zum Einsatz kommen, während die letzten beiden Typen sich auf in das XLX-System eingebundene Fremdsysteme beziehen. Durch diese Einbindung werden insbesondere Technologiebarrieren abgebaut, die ansonsten die praktische Bearbeitung realitätsnaher Aufgaben anhand komplexer (kommerzieller) Softwaresysteme, deren Installation mit hohem Administrationsaufwand einhergeht, auf dem heimischen PC be- oder sogar verhindern. Ferner sei bemerkt, dass zu manchen Aufgabentypen bei Eingabe einer zugehörigen Aufgabe in das System auch eine Musterlösung erfasst wird, was eine automatisierte Vorkontrolle studentischer Lösungen ermöglicht.

Das XLX-System ist als Client-Server-System realisiert und wurde im Hinblick auf die Erfüllung folgender Anforderungen hin entwickelt: *Serverseitig* muss eine dynamische Webseiten-Generierung möglich sein, es muss Plattformunabhängigkeit der entwickelten Programme gewährleistet sein, ferner größtmögliche Kompatibilität des generierten HTML-Codes und schließlich die Möglichkeit einer Anbindung von Zusatzsystemen (z.B. Datenbanksysteme, XSLT-Prozessoren) für

den Übungsbetrieb. Des Weiteren muss eine tageszeitunabhängige Verfügbarkeit garantiert werden, und dieses alles unter Beachten von Sicherheitsvorkehrungen. *Clientseitig* sind dagegen Betriebssystemunabhängigkeit, Hardware-Unabhängigkeit und Standortunabhängigkeit gefordert. Darüber hinaus wurden bei der Entwicklung dieses Systems grundsätzlich kostengünstige, frei erhältliche Komponenten oder Werkzeuge gegenüber kommerziellen Produkten bevorzugt. Insgesamt hat das System die in Abbildung 12-6 dargestellte, dreischichtige Architektur, welche sich an der oben in Abbildung 12-3 gezeigten Aufteilung orientiert.

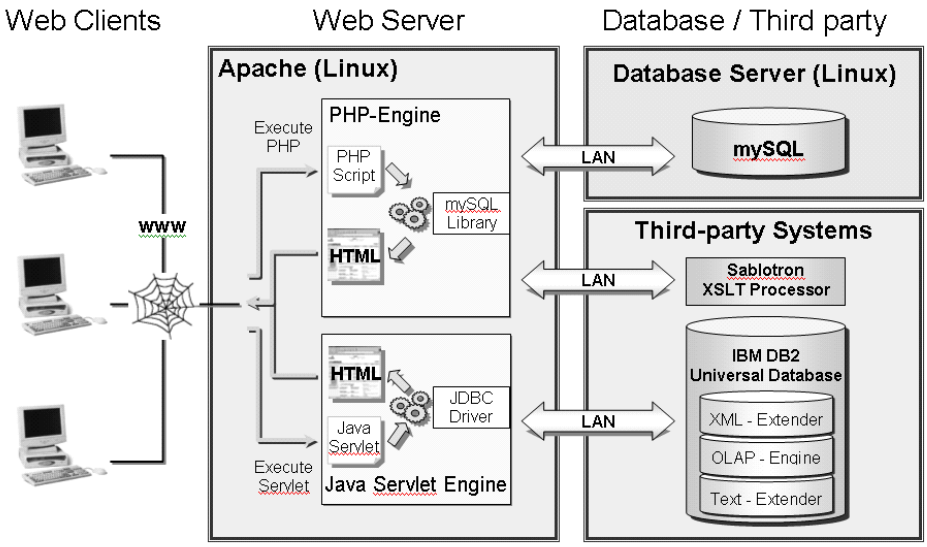


Abb. 12-6: *Architektur des XLX-Systems.*

Auf der äußersten Schicht befinden sich Web-Clients, die über einen Browser auf das System zugreifen können. Auf der mittleren Schicht befindet sich der Web-Server, der Client-Requests ausführt. Auf der innersten Schicht befindet sich ein Datenbank-Server, auf dessen Funktionalität bei vielen Aufgaben zurückgegriffen wird.

Der internetbasierte Zugriff auf das XLX-System über einen beliebigen Web-Browser ermöglicht einen flexiblen Zugang zum Übungssystem. Hierbei wird weitestgehende Unabhängigkeit von clientseitiger Hardware und Software gewährleistet und die zeitunabhängige Verfügbarkeit des XLX-Systems umgesetzt. Die vom System generierten HTML-Seiten können von allen gängigen Browsern dargestellt werden, wobei zur Identifikation und der Verfolgung einzelner Benutzer die Aktivierung von Cookies erforderlich ist. Ferner ist zur Realisierung eines größeren Eingabekomforts in einzelnen Bereichen des Systems die Aktivierung von JavaScript hilfreich, jedoch nicht zwingend erforderlich, insbesondere sind zur Benutzung der Plattform keine Browser-Erweiterungen (z.B. Plug-ins) notwendig.

Die Anwendungslogik des Systems ist ohne Ausnahme serverseitig implementiert. Somit dient der Browser nur als grafisches User-Interface für die XLX-Plattform, und es bedarf keiner besonderen Leistungsanforderungen an die Hardware des Clients. Für den orts- und zeitunabhängigen Übungsbetrieb außerhalb der Universität ist eine Internetverbindung notwendig, an die jedoch keine besonderen Geschwindigkeitsanforderungen bestehen. Die Middleware des Systems wird durch einen Linux-basierten Apache-Web-Server realisiert [Apa, Lin]. Zur Anmeldung an das System und zur weiteren Kommunikation wird eine SSL-gesicherte HTTP-Verbindung (HTTPS) zum Client verwendet. Auf diese Weise wird ein Basisschutz gegen Missbrauch und Sicherheit bei der Übermittlung vertraulicher Informationen (z.B. von Kennwörtern) gewährleistet.

Die dynamische Generierung der Webseiten geschieht entweder über die Ausführung von PHP-Skripten oder Java Servlets. Beide Technologien werden über die modulare Architektur des Apache-Web-Servers in das System eingebunden. Die Verwendung von Java Servlets dient der Anbindung des Übungssystems an das im XLX-System verwendete Datenbanksystem über JDBC. Auf dem Datenbank-Server sind in erster Linie relationale und objektrelationale Übungsdatenbanken zur Lösung datenbankspezifischer Aufgaben implementiert. Außerdem bietet die Funktionalität der Datenbank Möglichkeiten zur Anbindung einer OLAP-Engine (für *Online Analytical Processing*, vgl. [Voss00]) und unterschiedlicher Erweiterungen (für XML-, Text-, Video-, Image-Daten usw.). Über PHP wird zusätzlich auf eine lokale MySQL-Datenbank zurückgegriffen, die unter anderem benutzerspezifische Daten, Aufgaben und Lösungsabgaben für den Übungsbetrieb speichert. Außerdem erfolgt über PHP der Aufruf der Sablotron-XSLT-Bibliothek für den XML-Übungsbereich [Gin]. Weitere Einzelheiten zu XLX findet man bei [VoHL01].

12.4.5 Kommerzielle LMS

Zu kommerziell verfügbaren Learning-Management-Systemen finden sich im Web verschiedene Übersichten, z.B. die bereits genannten [wbt1, wbt2]. Darüber hinaus findet man Informationen hierzu bei verschiedenen elektronischen Magazinen zum Thema, etwa dem [elhub], dem [ElM] oder dem [OLM]; man vergleiche ferner [DEC], [EJP] sowie [WLE].

12.5 Zusammenfassung und Ausblick

Computerunterstützung von Lehr- und Lernvorgängen wird seit vielen Jahren aus Sicht der Forschung untersucht und kommerziell verwirklicht. Die Entwicklung in diesem Bereich begann mit einer Verwendung von Computern im Unterricht, etwa zu Präsentations- oder Demonstrationszwecken, und hat speziell in der zweiten Hälfte der 90er Jahre immer stärker das Internet sowie das World Wide Web einbezogen. Mittlerweile sind auch zahlreiche Projekte und Aktivitäten, die sich

mit der Nutzung neuer, insbesondere elektronischer Medien für das Lehren und Lernen befassen, in der Informatik-Literatur dokumentiert, wobei festzuhalten ist, dass andere als technische Sichtweisen auf das elektronische Lernen (beispielsweise pädagogisch-didaktische) zum Teil bereits eine längere Historie haben; an dieser Stelle sei z.B. auf [SLL] verwiesen.

Wir haben versucht, die wesentlichen Aspekte des Web-basierten Lernens (aus der Sicht der Informatik) im Überblick darzustellen. Dabei sind wir von einer Unterteilung einer Lernplattform in Autorensystem, Learning Management System und Laufzeitsystem ausgegangen und haben uns durchgehend daran orientiert. Es ergeben sich zahlreiche Anforderungen an solche Systeme, und es wird eine Reihe von Konzepten zu ihrer Realisierung eingesetzt.

Man kann davon ausgehen, dass das virtuelle, Web-basierte Lernen in den kommenden Jahren unsere Ausbildungsstätten, aber auch die arbeitende Gesellschaft immer stärker durchdringen und es hierfür zahlreiche weitere Anwendungen geben wird. So beschreiben z.B. [DaLH01] ein automatisiertes Werkzeug zur Transformation von Produktbeschreibungen in multimediale Trainingsanleitungen. Andere Überlegungen zielen auf virtuelles Einarbeiten von neuen Mitarbeitern in die Firmenprozesse ab. Das etwa von [CoTo01] oder [Pfah01] untersuchte kollaborative Lernen im virtuellen Schulungsraum wird weitere Verbreitung finden.

Andererseits zeigen die Erfahrungen mit Web-basierten Lernsystemen auch ihre Grenzen auf; hier sei der Leser auf die »Confessions of an E-Learning Dropout« von A. Rossett hingewiesen, wo in amüsanter Weise dargestellt wird, wie der an sich engagierte Lerner erst den Kurs etwas schleifen lässt, dann immer andere Vorwände findet, weshalb andere Dinge (auch am Computer bzw. im Web) wichtiger sind, und schließlich den Faden verliert. Es wird heute in vielen Quellen davon ausgegangen, dass man in Zukunft das »Blended Learning« häufiger als das reine E-Learning antreffen wird. Dabei wird rein elektronisches Lernen mit Lernen in traditionellen Präsenzveranstaltungen gemischt, so dass es Teilnehmern wieder regelmäßiger möglich wird, mit anderen Lernenden zusammenzutreffen.

Wir wollen abschließend zwei Bereiche ansprechen, in denen sich aus unserer Sicht in den kommenden Jahren auf konzeptioneller Ebene neue Entwicklungen vollziehen werden: Einerseits wird das elektronische Lernen neue *Geschäftsmodelle* hervorbringen bzw. benötigen. So wird z.B. im Rahmen der erwähnten w3l-Plattform ein Modell entwickelt, bei welchem jeder Autor Inhalte in das System einbringen kann; werden diese von Lernenden abgerufen, so wird der Autor hierfür im Rahmen einer Tantiemenregelung entschädigt. Ferner ist denkbar, dass demnächst spezielle Intermediäre passende Lehrangebote an Lernende vermitteln, oder dass Recommender-Systeme bestimmte Lerneinheiten individuell empfehlen. Hierzu wird dann eine neuartige Form der Zertifizierung von Lerninhalten erforderlich werden.

Andererseits glauben wir auch an eine zunehmende *Workflow-Unterstützung* des elektronischen Lernens. Neben der Datenbankunterstützung einer Web-basierten Lernplattform, bei der es ja um die Erstellung, die Speicherung und die

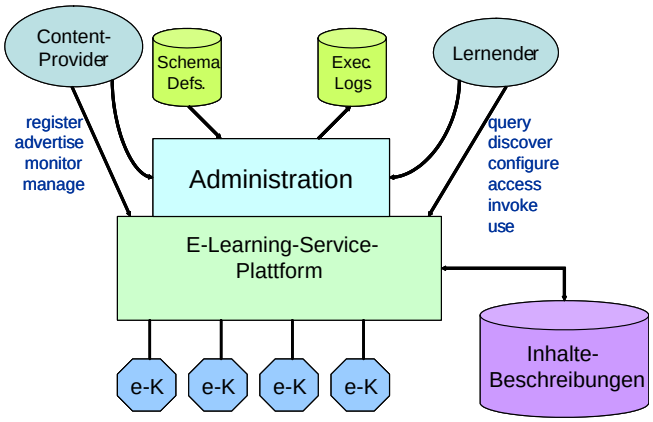


Abb. 12-7: *E-Learning als Web Service*

Verwaltung von Inhaltsinformation geht, ist es zunehmend von Bedeutung, sich auch über die einzelnen, in einem Lernsystem vorkommenden Abläufe, deren Zusammenspiel und ihre Definier- sowie Änderbarkeit Gedanken zu machen. Grundsätzlich sind sowohl Lernprozesse als auch administrative Vorgänge (und sogar Autorensitzungen) in einer solchen Form (als Prozess oder als Workflow) modellierbar, was wir weiter oben im Zusammenhang mit XLX bereits kurz angedeutet haben. Hierdurch werden mehrere Ziele erreicht: Einerseits werden Akteure, die von ihnen ausgeführten Tätigkeiten und die im Rahmen dieser Tätigkeiten bearbeiteten Objekte in Beziehung zueinander gesetzt. Man kann dann z.B. festlegen, welche Bearbeitungsreihenfolgen zum Absolvieren eines bestimmten Kurses möglich sind, wie Tutoren vorgehen sollten oder welche Tätigkeiten ein Administrator regelmäßig durchzuführen hat. Man kann aber im Rahmen einer flexiblen Modellierung auch dynamische Änderungen erfassen oder zulassen, so dass sich komplexere Abläufe aus einfacheren zusammensetzen. So kann z.B. ein Kurs aus einem einführenden Teil bestehen, bei welchem Material zu lesen ist, gefolgt von einem praktischen Teil mit Fallstudien, gefolgt von einem repetitorischen Teil mit Tests, gefolgt von einem weiterführenden Teil mit zu lesendem Material; alle Teile können einzeln als Workflows modelliert werden und dann in der genannten oder in anderen Zusammensetzungen verwendet werden.

Werden Geschäftsmodelle und Prozessorientierung angemessen integriert, ist es vorstellbar, dass E-Learning demnächst als ein *Web Service* zur Verfügung gestellt wird; dies haben wir in Abbildung 12-7 skizziert. Der E-Learning-Web-Service würde dabei über eine spezielle Plattform abgewickelt, welche elektronische Kurseinheiten und deren Benutzer verwaltet. In ganz ähnlicher Weise wie man heute E-Services zur Abwicklung von Geschäften und Business-Transaktionen entwickelt, vgl. etwa [PiTs01], können Content-Provider die von ihnen angebotenen Lerneinheiten beim Service-Provider registrieren, ankündigen, verwalten und deren Verwendung überwachen. Lernende dagegen können die Plattform anfragen und nach benötigten »Lern-Diensten« durchsuchen, Kurse nach eigenen

Bedürfnissen konfigurieren, auf Inhalte zugreifen, Dienste aufrufen und benutzen. Die Plattform verwaltet Kursangebote und Lernendeninformation; eine Administrationskomponente überwacht die Ausführungen und die Ablage von Inhalten.

Literatur

[AdCP02] Adelsberger, H.H., B. Collis, J.M. Pawlowski, Hrsg.. *Handbook on Information Technologies for Education and Training*. Springer-Verlag, Berlin, 2002.

[ADL01] ADL *The SCORM Content Aggregation Model*, Version 1.2. Advanced Distributed Learning Initiative, October 2001.

[AnRi01] Anido-Rifon, L., et al. *A Component Model for Standardized Web-Based Education*. ACM Journal of Educational Resources in Computing, 1(2), Summer 2001.

[Barr01] Barritt, C. *CISCO Systems Reusable Learning Object Strategy – Designing Information and Learning Objects Through Concept, Fact, Procedure, Process, and Principle Templates*, Version 4.0. White Paper, CISCO Systems, Inc., November 2001.

[Bart01] Barthelmeß, H. *ViKar-Campus: Chancen und Risiken*. Proc. Learntec 2001.

[Bein01] Beinhauer, M. *Vom Projekt zur virtuelle Universität – Erfahrungsbericht WINFOLine und Virtuelle Saar-Universität*. Proc. Learntec 2001.

[Cham99] Chamberlin, D. *DB2 Universal Database – Der unentbehrliche Begleiter.* Addison-Wesley, Bonn, 1999.

[CoGr01 Coners, A., H.L. Grob. *Konstruktion und Nutzung von MultiMediaModellen mit model.cube*. Verlag Franz Vahlen, München 2001.

[CoTo01] Constantini, F., C. Toinard. *Collaborative Learning with the Distributed Building Site Metaphor.* IEEE Multimedia, Issue 3, pp. 21-29, 2001.

[DaLH01] Day, Y.F., P. Liu, L.H. Hsu. *Transforming Large-Scale Product Documents into Multimedia Training Manuals*. IEEE MultiMedia, July issue, pp. 39-45, 2001.

[DeBr99] De Bra. *Design Issues in Adaptive Web-Site Development*. In Proc. 2nd Workshop on Adaptive Systems and User Modeling on the WWW, 1999.

[DeCa98a] De Bra, P., L. Calvi. *AHA! An open Adaptive Hypermedia Architecture*. The New Review of Hypermedia and Multimedia 4, Taylor Graham Publishers, pp. 125-129, 1998.

[DeCa98b] De Bra, P., L. Calvi. *AHA: a Generic Adaptive Hypermedia System*. In: Proc. 2nd Workshop on Adaptive Hypertext and Hypermedia, Pittsburgh, pp. 5-12, 1998.

[Ditt02] Dittler, U., Hrsg. *E-Learning - Erfolgsfaktoren und Einsatzkonzepte mit interaktiven Medien*. R. Oldenbourg-Verlag, München, 2002.

[Dobe00] Doberkat, E.-E., et al., Hrsg. *Multimedia in der wirtschaftswissenschaftlichen Lehre – Erfahrungsbericht*. LIT Verlag, Münster, 2000.

[Down01] Downes, S. *Learning Objects: Resources for Distances Education Worldwide*. International Review of Research in Open and Distance Learning 2 (1), 2001.

[Ecks99] Eckstein, R. (1999). *XML Pocket Reference*. O'Reilly & Associates, Sebastopol, CA, 1999.

[EfHo98] Effelsberg, W., C. Hornung. *Lehren und Lernen im Internet*. Informationstechnik und Technische Informatik 40 (2), 16-22, 1998.

[Ehre01] Ehrenberg, D., et al. *Implementierung von interuniversitären Lehr- und Lernkooperationen: Das Beispiel WINFOLine*. Wirtschaftsinformatik 43 (2001) 1, 5-12.

[ElFS01] El Saddik, A., S. Fischer, R. Steinmetz. *Reusability and Adaptability of Interactive Resources in Web-Based Educational Systems*. ACM Journal of Educational Resources in Computing 1 (1), 2001.

[FHSU00] Ferst, O.K., K. Hahn, K. Schmitz, C. Ullrich. *Funktionen und Architektur einer Internet-Lernumgebung für individuelles und kooperatives Lernen*. In: Uellner, S., V. Wulf (Hrsg.), Vernetztes Lernen mit digitalen Medien. Physica-Verlag, Heidelberg, 53-68, 2000.

[FeSc01] Ferstl., O.K., K. Schmitz. *Integrierte Lernumgebungen für virtuelle Hochschulen*. Wirtschaftsinformatik 43 (2001) 1, 12-22.

[Fisc01] Fischer, S. *Course and Exercise Sequencing Using Metadata in Adaptive Hypermedia Learning Systems*. ACM Journal of Educational Resources in Computing, 1(1), Spring 2001.

[Grob00] Grob, H.L. *Das Konzept der Wirtschaftswissenschaftlichen Fakultät der Westfälischen Wilhelms-Universität Münster.* In: [Dobe00], 57- 127.

[Grob01] Grob, H.L., Hrsg. *cHL computergestützte Hochschullehre – Dokumentation zum cHL-Tag 2000*. Lit-Verlag, Münster, 2001.

[GuOt97] Gunzenhäuser, R., M. Otto. *Grundlagen intelligenter Lehr- und Lernsysteme*. Informationstechnik und Technische Informatik 39 (6), 12-15, 1997.

[Haak01] Haake, J.M. *Erlernen kooperativer Arbeitsprozesse in einem virtuellen Lernraum*. In W. Brauer, Th. Mück, Hrsg., Informatik 2001, Tagungsband der GI/ÖCG-Jahrestagung, 1223-1230, 2001.

[Hamp01] Hampel, Th., et al. *»Ein Schulmeister muss singen können« – Die drei Säulen der Paderborner DISCO*. Wirtschaftsinformatik 43 (2001) 1, 69-76.

[Hara99] Harasim, L. *A Framework for Online Learning: The Virtual-U*. IEEE Computer, September 1999, pp. 44-49.

[HeNe99] Henze, N., W. Nejdl. *Adaptivity in the KBS Hyperbook System*. In Proc. 2nd Workshop on Adaptive Systems and User Modeling on the WWW, 1999.

[HSKV01] Hilt, V., C. Schremmer, C. Kuhmünch, J. Vogel. *Erzeugung und Verwendung multimedialer Teachware im synchronen und asynchronen Teleteaching*. Wirtschaftsinformatik 43 (2001) 1, 23-33.

[IEEE02] IEEE Standards Department. *Draft Standard for Learning Object Metadata*. IEEE Publication P1484.12.1/D6.4, March 2002.

[IMS01] IMS Global Learning Consortium, Inc. *IMS Content Packaging Best Practice Guide*, Version 1.1.2. August 2001.

[Kand99] Kandzia, P.-Th. *VIROR – Die virtuelle Hochschule Oberrhein*. In Proc. 29. GI-Jahrestagung, Paderborn, 1999.

[Kay00] Kay, M. *XSLT Programmer's Reference*. Wrox Press, Birmingham, UK, 2001.

[Koop01] Kooper, R. *Modeling units of study from a pedagogical perspective – the pedagogical meta-model behind EML*. Open University o the Netherlands, Heerlen, The Netherlands, June 2001.

[Leid01] Leidig, T. *L^3 – Towards an Open Learning Environment*. ACM Journal of Educational Resources in Computing, 1(1), Spring 2001.

[LeOb01] Lenz, K., A. Oberweis. *Interorganizational Business Process Management with XML-Nets*. In Proc. 2nd Int. Colloquium on Petri Net Technologies for Modelling Communication Based Systems, Berlin, Sept. 2001, 139-156.

[MüOt99] Müller, R., Th. Ottmann. *The «Authoring on the Fly« System for Automated Recording and Replay of (Tele) Presentations*. ACM/Springer Multimedia Systems 8 (3), 1999.

[Pfah01] Pfahl, D. et al. *CORONET-Train: A Methodology for Web-Based Collaborative Learning in Software Organizations*. Techn. Report, Fraunhofer IESE, 2001.

[PiTs01] Pilioura, T., A. Tsalgatidou. *E-Services: Current Technology and Open Issues*. In Proc. 2nd International Workshop on Technologies for E-Services, Rome, Springer-Verlag, Berlin, pp.1-15, 2001.
[QuNe00] Qu C., W. Nejdl: *Bridging O-Telos and XML with XML Schema: The Authoring Environment for KBS Adaptive Hyperbook*. Technical Report, University of Hannover, 2000.
[Ross02] Rossett, A., Hrsg. *The ASTD E-Learning Handbook*. McGraw-Hill, New York, 2002.
[Schr00] Schremmer, C., et al. *Erfahrungen mit synchronen und asynchronen Lernszenarien an der Universität Mannheim*. PIK 23 (2000) 3, pp. 121-128.
[ShSC01] Shang, Y., H. Shi, S.-S. Chen. *An Intelligent Distributed Environment for Active Learning*. ACM Journal of Educational Resources in Computing, 1(2), Summer 2001.
[SiKS02] Silberschatz, A., H.F. Korth, S. Sudarshan. *Database System Concepts*, 4th edition. McGraw-Hill, Boston, MA, 2002.
[Simo01] Simon, B. *E-Learning an Hochschulen*. Josef Eul Verlag, Lohmar, 2001.
[SüFr01] Süß, C., B. Freitag. *Learning Material Markup Language LMML*. IFIS-Report 2001/03, Universität Passau,2001.
[TrLi01] Trahasch S., J. Lienhard: *Hochschulübergreifende Wiederverwendung digitaler Lehrmodule*. In K. Bauknecht, W. Brauer, Th. Mück, Hrsg., Informatik 2001, Tagungsband der GI/ÖCG-Jahrestagung, Band II, pp. 1260-1264, 2001.
[Voss00] Vossen, G. *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*, 4. Auflage. R. Oldenbourg Verlag, München, 2000.
[VoHL01] Vossen, G., B. Hüsemann, J. Lechtenbörger: *XLX – Eine Lernplattform für den universitären Übungsbetrieb*. Arbeitsbericht Nr. 79, Institut für Wirtschaftsinformatik, Universität Münster, 2001.
[WeVo02] Weikum, G., G. Vossen. *Transactional Information Systems – Theory, Algorithms, and The Practice of Concurrency Control and Recovery*. Morgan Kaufmann Publishers, San Francisco, CA, 2002.
[Wile00] Wiley, D.A. *Learning Object Design and Sequencing Theory*. Ph.D. dissertation, Brigham Young University, 2000.
[WuHD99] Wu, H., G.J. Houben, P. De Bra. *Supporting User Adaptation in Adaptive Hypermedia Applications*. Technical Report, Dept. of Computing Science, TU Eindhoven, 1999.

Ausgewählte Web-Links

[adl] Advanced Distributed Learning Network (ADLNet), http://www.adlnet.org/
[aicc] Aviation Industry CBT Committee, http://www.aicc.org/
[Apa] Apache Software Foundation, http://www.apache.org/
[c2t2] Comparative Analysis of Online Educational Delivery Applications, http:// www.c2t2.ca/landonline/
[CaS] Open Source-Software CampusSource, http://www.campussource.de
[dcmi] Dublin Core Metadata Initiative, http://dublincore.org/
[DEC] Distance Education Clearinghouse, http://www.uwex.edu/disted/home.html
[elhub] The E-Learning Hub, http://www.e-learninghub.com/index.html
[ElM] E-learning Magazine, http://www.elearningmag.com/
[EJP] The New eLearning Jump Page, http://www.internettime.com/e.htm
[ets] Educational Testing Service, http://www.ets.org/

[FHA] Fachhochschule Augsburg, Verzeichnis multimedialer Online-Lehrveranstaltungen, http://www.fh-augsburg.de/informatik/projekte/mebib/vom1.html
[Gin] Ginger Alliance, http://www.gingerall.com/
[hot] Hot Potatoes Half-Baked Software, http://web.uvic.ca/hrd/halfbaked/
[ieee] IEEE Distance Learning Campus, http://www.computer.org/distancelearning
[ims] IMS Global Learning Consortium, Inc., http://www.imsproject.org/
[isst] Fraunhofer-Institut Software- und Systemtechnik, Berlin und Dortmund, http:// panda.isst.fhg.de:8080/CourseDesigner.html
[Lin] Linux HeadQuarters, http://www.linuxhq.com/
[LOT] Eduworks Corp., Learning Object Tutorial, http://www.eduworks.com/LOTT/ tutorial/index.html
[ltsc] IEEE Learning Technology Standards Committee, http://ltsc.ieee.org/
[lrn] Microsoft Learning Resource iNterchanges (LRN) Toolkit, Version3.0, http:// www.microsoft.com/elearn
[mar] Comparison of Online Course Delivery Software Products, http://www.marshall. edu/it/cit/webct/compare/comparison.html
[OLM] OnlineLearning Magazine, http://www.onlinelearningmag.com
[ready] ReadyGo Web Course Builder, http://www.readygo.com/
[SLL] Stanford University Learning Laboratory, http://sll.stanford.edu
[vcrp] Virtueller Campus Rheinland-Pfalz, http://www.vcrp.de/
[vhb] Virtuelle Hochschule Bayern, http://www.vhb.org
[vhbw] Virtuelle Hochschule Baden-Württemberg, http://www.virtuelle-hochschule.de
[ViKar] Virtueller Hochschulverbund Karlsruhe, http://www.vikar.de
[VIROR] Virtuelle Hochschule Oberrhein, http://www.viror.de
[visum] VISUM: Virtuelles System zum Unterricht in Mathematik, http://visum.uni-muenster.de
[vgu] Virtual Global University, http://www.vg-u.de
[wbt1] WBTIC: WBT Tools Resources, http://www.filename.com/wbt/pages/wbttools.htm
[wbt2] Web Based Learning Resources Library, http://web.ce.utk.edu/weblearning/, siehe auch http://www.outreach.utk.edu/weblearning/
[xlx] *Extreme eLearning Experience* der Universität Münster, https://dbms.uni-muenster. de/xlx
[w3l] w3l GmbH, Lebenslanges Lernen im Web, http://www.swt.ruhr.uni-bochum.de/ w3l
[WLE] WWW Learning Environments, http://www.oc.utwente.nl/w3ls/english/wwwvb. htm

13 Kommerzielle Systeme zur Speicherung, Verwaltung und Anfrage von XML-Dokumenten

Holger Meyer, Meike Klettke

Kurzfassung

In diesem Kapitel wird ein Überblick über verschiedene Systeme gegeben, die XML-Dokumente dauerhaft und sicher speichern und anfragen können. Dazu werden die Erweiterungen von objektrelationalen Datenbanksystemen vorgestellt, die notwendig sind, um XML-Dokumente zu speichern und anzufragen. Weiterhin werden XML-Server dargestellt, die speziell für die Speicherung und Anfragen von XML-Daten entwickelt wurden. Zahlreiche Systeme werden in dem Kapitel beschrieben, und die Merkmale der XML-Speicherung, Indexierung und Anfrage werden vergleichend dargestellt.

13.1 Einführung

Die Forderung einer Vielzahl industrieller Anwendungen, XML in erheblichem Umfang zu speichern, anzufragen und weiter zu verarbeiten, führte zur Entwicklung kommerzieller XML-Verarbeitungssoftware. Die Möglichkeiten zur XML-Verwaltung soll an ausgewählten Beispielen in Datenbanksystemen genauso wie in dedizierten XML-Servern untersucht werden.

Neben IBM mit DB2 (Abschnitt 13.2) bieten auch Oracle 9i (Abschnitt 13.3) und der Microsoft SQL Server 2000 (Abschnitt 13.4) die Speicherung mit einem XML-Datentyp sowie eine Abbildung von XML-Daten auf relationale oder objektrelationale Strukturen. Der Export von Anfrageergebnissen in XML-Form wird ebenfalls geboten. XML dient dabei als Behälter relationaler oder objektorientierter Strukturen. Seine eigenen Darstellungskonzepte bleiben eher ungenutzt.

Das Datenmodell ist bei diesen traditionellen Datenbanksystemen das relationale oder objektrelationale Modell, die Anfragesprache weiterhin SQL. Wenn auch die direkte Speicherung von XML-Fragmenten oder ganzen Dokumenten in einem XML-Datentyp unterstützt wird und auch XPath-Anfragemöglichkeiten geboten werden, sind diese jedoch über Spracherweiterungen, neue Datentypen und zugehörige Funktionen von SQL realisiert.

Ein System, das auf dem objektorientierten Datenmodell der ODMG basiert, ist *POET*. Hier erfolgt eine Abbildung von XML über Java-Klassen auf objektorientierte Strukturen, die mit den Mitteln von OQL abfragbar sind. Weitergehende Informationen sind in Abschnitt 13.8 zu finden.

Eine Klasse von weiteren Systemen verwendet XML direkt als logisches Datenmodell, auch wenn sie intern durchaus auf Datenbanksysteme zur Verwaltung aufsetzen. Diese Systeme werden oftmals als XML-Server, teilweise auch als native XML-Datenbanksysteme, bezeichnet. Object Designs *eXcelon* (Abschnitt 13.5) bildet XML auf Objekte ab und erlaubt Anfragen mit XPath ähnlich wie *Tamino* der Software AG (Abschnitt 13.6), das wohl intern ein Netzwerk- oder hierarchisches Modell verwendet und mittlerweile neben XPath auch XQuery als Anfragesprache unterstützt. Neu entwickelte oder gezielt auf XML angepasste Speicherungstechniken verwendet der aus dem universitären Umfeld stammende XML-Server *Infonyte* (Abschnitt 13.7).

Zu nativen XML-Datenbanksystemen gehören meistens Werkzeuge zur Definition von XML-Schemata und XML-Editoren zur Erstellung von XML-Dokumenten. Eine Integration mit Transformationssprachen wie XSLT ist eine weitere Eigenschaft dieser Systeme.

13.2 DB2 mit XML- und TextExtender

Das objektrelationale Datenbanksystem DB2 Universal Database von IBM [Cham98] bietet vielfältige Unterstützung für die Speicherung von XML. Mittels des Extender-Mechanismus, der es gestattet, eigene Erweiterungen in das Datenbanksystem einzubinden, wurden Speicherungstechniken und Anfragemechanismen für Volltexte und XML-Dokumente in DB2 integriert.

DB2 Universal Database (UDB) kann XML in drei Formen verwalten:

1. durch direkte Speicherung in einem XML-Datentyp, der eine Manipulation und Anfragen mit spezifischen Methoden unterstützt,
2. mittels Abbildung von XML- auf objektrelationale Strukturen und die Darstellung von Anfrageergebnissen als XML-Strukturen sowie
3. durch die Speicherung als Volltext und die Realisierung von Information-Retrieval-Operationen unter Beachtung der XML-Dokumentstruktur.
4. Die ersten beiden Techniken werden vom XML-Extender, die dritte vom TextExtender umgesetzt.

Darüber hinaus gibt es eine Menge an Werkzeugen und Zusätzen, die ebenfalls eine XML-Verarbeitung mit DB2 zum Ziel haben. Genannt seien *XPERANTO* zum Anfragen und Export über XML-Sichten und die Integration von DB2 in die Web-Server-Umgebung *WebSphere* und die zugehörige Skriptsprache *Net.Data*.

13.2.1 XML-Extender – Überblick

Der XML-Extender vereinigt objektrelationale Datenbanktechnologie und XML. Er ist integraler Bestandteil ab *DB2 UDB* V7.1 und für viele Plattformen verfügbar: *IBM AIX*, *Windows NT*, *Linux*, *SUN Solaris* sowie für *AS/400* und *OS/390*.

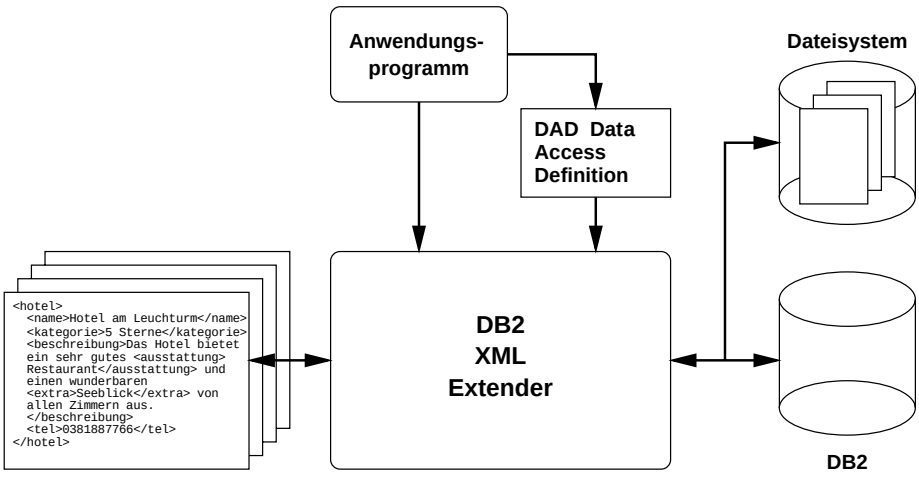


Abb. 13-1: *Überblick XML-Extender*

Realisiert werden vom XML-Extender zwei Speicherungstechniken:

- Die XML Collection stellt einen Mechanismus zur Abbildung auf objektrelationale Strukturen dar.
- XML Column ist ein XML-Datentyp und zur Speicherung von ganzen XML-Dokumenten oder -Fragmenten einsetzbar.

Die genaue Form der Speicherung wird mit einer Data Access Definition, kurz DAD genannt, beschrieben. Die DAD kann dabei sowohl die Abbildung beim Importvorgang als auch die zu exportierenden XML-Strukturen spezifizieren. Zur Suche in den gespeicherten Dokumenten kommen die Anfragemöglichkeiten von SQL zum Einsatz, und in Methoden der Extender auch XPath-Ausdrücke. Der TextExtender erlaubt Information Retrieval nach den in SQL/MM Fulltext festgelegten SQL-Spracherweiterungen.

Einen Überblick zum DB2 XML-Extender gibt Abbildung 13-1. Anwendungsprogramme interagieren direkt mit DB2 über die SQL-Anfragesprache und Aufrufen von Stored Procedures und nutzerdefinierten Funktionen (UDFs) des XML-Extenders und beschreiben die gewünschten Abbildungsvorgänge beim Import und Export von XML-Dokumenten mit DAD-Dateien. Die Ablage von XML kann direkt in DB2 erfolgen. Dokumente können aber auch extern im Dateisystem abgelegt sein, während nur bestimmte Dokumentbestandteile in DB2 gespeichert und indexiert werden. Die Integration des Dateisystems und die Über-

wachung der Zugriffe erfolgt dann mit dem so genannten DataLink-Mechanismus.

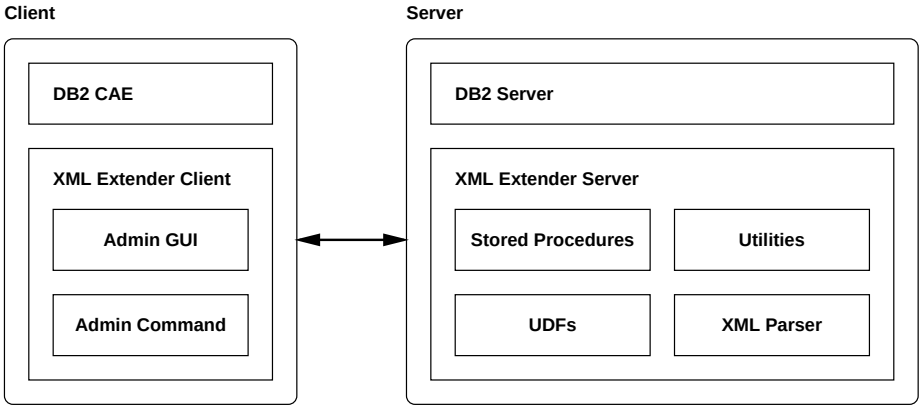


Abb. 13-2: *XML-Extender-Komponenten*

Der XML-Extender ist mit einem Erweiterungsmechanismus von DB2, der Extender-Technologie, implementiert worden. Diese erlaubt, neue nutzerdefinierte Datentypen (UDTs) und nutzerdefinierte Funktionen, Stored Procedures, Indexierungstechniken und weitere Hilfsfunktionen zu einer Erweiterung, einem Extender, zusammenzufassen. Zu diesen Komponenten auf Datenbank-Server-Seite, die Abbildung 13-2 darstellt, kommen neben der Extender-Registrierung und allgemeinen Einstellungen über den DB2-Administrator-Client (CAE) ein spezieller XML-Extender-Client, der u.a. Mechanismen zur Arbeit mit DAD-Dateien bereitstellt.

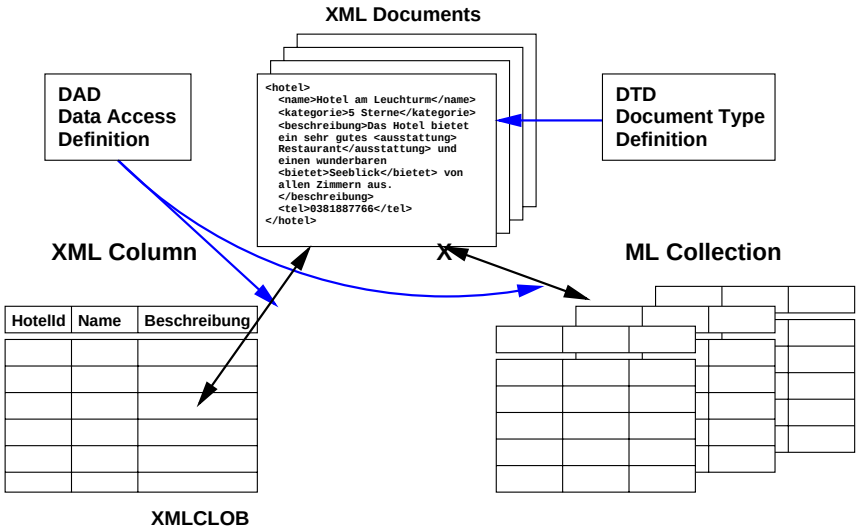


Abb. 13-3: *Beschreibung der XML-Speicherung mit DAD-Datei*

13.2.2 Speicherungs- und Zugriffstechniken

Bevor die DAD-Datei zur Steuerung des Abbildungsprozesses von XML-Dokumenten auf Datenbankinhalte und umgekehrt vorgestellt wird, sollen die DB2-Techniken zur Speicherung und für den Zugriff betrachtet werden. In Abbildung 13-3 wird das Zusammenwirken von DAD und den Speicherungsformen zusammengefasst.

Die Speicherungsformen und zugehörigen Anfragemechanismen sind:

- *XML Column* stellt einen XML-Datentyp bereit. Dieser nutzerdefinierte Datentyp kann als Typ beliebiger Datenbankattribute eingesetzt werden. Begleitet wird dieser Datentyp von nutzerdefinierten Funktionen für den Zugriff auf und die Manipulation von gespeicherten Werten.
- *XML Collection* erlaubt eine relationale Speicherung von XML. Der Zugriff und die Manipulation erfolgt direkt mit SQL und Stored Procedures.

Darüber hinaus kann der *DB2 TextExtender* u. a. auch XML-Dokumente speichern und indexieren. Er bietet die für das Information Retrieval übliche Volltextfunktionalität an.

XML-Datentyp XML Column

XML Column stellt einen abstrakten Datentyp dar, dessen Struktur über einen UDT (User Defined Type) definiert und dessen Verhalten über UDFs (User Defindes Functions) implementiert wird. Eine Applikation kann ihn zur Speicherung als Datentyp von Datenbankattributen anwenden. Er besitzt folgende Eigenschaften:

- Der Datentyp verwendet das XPath-Datenmodell in Form von Knotenmengen zur Abbildung von XML-Strukturen auf Tabellenstrukturen.
- Die Identifikation von mehrfach auftretenden Elementen erfolgt über Sequenznummern, die die Dokumentordnung abbilden.
- XPath-basierte Anfragen an Einzeldokumente sind mit UDFs möglich.
- Es gibt weiterhin Update-Methoden für Elemente und Attribute mittels XPath-Ausdrücken.
- Für eine schnelle Suche kann eine Indexierung einzelner Elementinhalte oder Attributwerte über Zusatztabellen (Werteindexe), so genannte Side Tables erfolgen. Die Angabe zu indexierender Elemente und Attribute erfolgt über Pfadangaben verbunden mit der Angabe des verwendenden SQL-Datentyps in der DAD-Datei.

Von XML Column werden genau genommen drei UDTs bereitgestellt, die sich in der konkreten Ablageform des XML-Dokumentes unterscheiden. Sie stellen die Struktur eines abstrakten Datentyps (ADT) dar.

- `XMLFile` stellt nur eine Referenz auf eine externe XML-Datei dar, die über den DB2-DataLink-Mechanismus verwaltet wird.
- `XMLVarchar` ist für interne, kurze Dokumente geeignet.
- `XMLCLOB` kann für interne, große Dokumente benutzt werden.

Das Verhalten des abstrakten Datentyps wird durch UDFs umgesetzt. Es gibt Funktionen und Methoden für den Import, das Retrieval, für Update-Operationen und die Extraktion von XML-Bestandteilen.

So erlauben die `Extract`-Funktionen die Extraktion von XML Element- und Attributwerten aus Dokumenten. Dabei wird eine Konvertierung von Zeichenfolgen aus XML-Dokumenten in Werte entsprechender SQL-Datentypen vorgenommen. Vom Extender werden pro SQL-Datentyp entsprechende Funktionen bereitgestellt, so extrahiert die Funktion `extractDouble` einen SQL-Fließkommawert.

Für jeden Datentyp gibt es zwei Formen der UDFs:

- Skalare Funktionen liefern jeweils nur einen Wert zurück.
- Tabellen-Funktionen, so genannte Table Functions, liefern eine Menge von (komplexen) Werten in Form einer Tabelle als Ergebnis.

Beispiel 13-1. Die Anwendung der extract-Funktionen soll an zwei Beispielen gezeigt werden. Eine erste Anfrage extrahiert die Zimmerpreise mit der skalaren Funktion.

```
select db2xml.extractDouble(zimmertyp, '/zimmertyp/preis')
from zimmertyp
where hname = 'Hotel am Leuchtturm'

select *
from table(db2xml.extractStrings(
  db2xml.XMLFile('/home/hme/hotel/hotels.xml'), '/hotel/*/name'))
  as hotelname
```

In der zweiten Anfrage werden alle Hotelnamen in Form einer Tabelle extrahiert. □

Abbildung auf Tabellen mit XML Collection

Unter dem Begriff XML Collection werden alle Methoden zur Abbildung von XML-Strukturen auf normale objektrelationale Strukturen und umgekehrt zusammengefasst. Im Einzelnen sind dies:

- die Dekomposition von XML-Dokumente in Tabellenstrukturen,
- die Speicherung der Elementinhalte und Attribute als Datenbankattribute und
- die direkte Abbildung von Relationen auf XML-Strukturen.

Die eigentliche Verarbeitung wird durch eine Menge von Stored Procedures realisiert, die alle Bezeichner mit dem Präfix `dxx` haben, wie etwa `dxxRetrieve`. Es gibt Prozeduren für verschiedene Aufgabenbereiche:

- die Zerlegung von XML-Dokumenten in neue oder existierende Tabellen mit `dxxShredXML` unter Verwendung einer DAD,
- die Generierung von XML-Dokumenten aus existierenden Daten oder zerlegten Dokumenten mit `dxxGenXML` und
- das Bereitstellen von Typspezifikation und Werten für dynamische Abbildungsvorschriften (DADs) mit `dxxRetrieve`.

13.2.3 Beschreibung der XML-Speicherung mit DAD

Die DAD-Datei, selbst in Form eines XML-Dokumentes definiert, beschreibt die Abbildung zwischen XML-Dokumenten und Datenbanktabellen. Es kann eine Abbildung sowohl auf spezielle Attribute vom Typ XML als auch auf Attribute erfolgen, die von einem SQL-Standarddatentyp sind. Erstere Form ist die XML-Column-, die zweite die XML-Collection-Speicherung.

Für die Auswahl und Extraktion von Dokumentbestandteilen wie Elementen und Attributen werden von der DAD-Datei Pfadausdrücke in Form von XPath-Ausdrücken in verkürzter Schreibweise verwandt.

Die Abbildung auf existierende Tabellenstrukturen in der XML Collection verwendet das XPath-Datenmodell mit Knotenmengen als Basis, um komplette XML-Strukturen in Tabelleninhalte zu überführen. In einer DAD-Datei finden zwei Abbildungsmethoden eine Anwendung:

- Mit SQL kann eine Komposition von XML-Dokumenten erfolgen.
- Ein `RDB_node` kann zur Komposition als auch zur Zerlegung von Dokumenten einen Einsatz finden.

Beide Methoden sollen vorgestellt werden.

SQL zur Komposition von XML-Dokumenten

Eine einfache Technik, um Anfrageergebnisse als XML-Dokumente auszugeben, ist die Angabe von `SQL_stmt`-Elementen in einer DAD-Datei. Mit der Angabe von SQL `select`-Anweisung kann eine Abbildung von Datenbankattributen auf XML-Daten als Attributwerte mit `attribut_node` und Elementinhalt mit `text_node` bestimmter XML-Elemente `element_node` erfolgen. Ein einfaches Beispiel wird in Beispiel 13-2 gezeigt.

Diese Technik hat eine Reihe von Einschränkungen, so ist etwa die Schachtelung von Elementinhalten oder die Extraktion von strukturierten Datenbankattributen nicht einfach umzusetzen.

Beispiel 13-2. Diese DAD-Datei demonstriert die Erzeugung von XML über die SQL-Methode. Das Element `SQL_stmt` enthält eine SQL-Anweisung zur Berechnung einer Relation. Die aus dem Ergebnis zu übernehmenden Werte werden mit dem Element `column_name` benannt und in das zu konstruierende Dokument übernommen.

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "/usr/lpp/db2/dxx/dtd/dad.dtd">
<DAD><validation>NO</validation>
<Xcollection>
  <SQL_stmt>
    select band.name, album.title, album.label, album.year
    from band, album
    where band.name = album.band
    order by band.name, album.title, album.year
```

```
  </SQL_stmt>
  <prolog>?xml version="1.0"?</prolog>
  <doctype>!DOCTYPE band-info SYSTEM "band-info.dtd"</doctype>
  <root_node>
    <element_node name="band">
      <attribute_node name="name">
        <column_name="band.name"/>
      </attribute_node>
      <element_node name="album">
        ...
    </element_node>
  <root_node>
</Xcollection>
</DAD>
```

Da unter dem root_node-Element für jedes Ergebnistupel ein Elementknoten band mit element_node erzeugt wird, sind keine Gruppenbildungen und tiefer geschachtelte Ergebnisdokumente möglich. □

RDB_node zur Komposition und Dekomposition

Im Gegensatz zur SQL-Methode verwendet RDB_node eine reine XML-Syntax. Genau genommen erfolgt eine Spezifikation von Tabellen und Beziehung zwischen Tabellen und XML-Elementen mit speziellen Elementen, den RDB_node-Knoten. Diese RDB_node-Elemente müssen sowohl für das Wurzelelement, also das auf oberster Ebene spezifizierte element_node-Element, als auch für alle Text- und Attributknoten in einer DAD-Datei angegeben werden. Pro RDB_node können referenzierte Tabellennamen, Spaltennamen und wahlweise Bedingungen für Attribut- (attribute_node) und Textknoten (text_node) angegeben werden. Das bei der SQL-Methode eingesetzte SQL_stmt-Element entfällt.

Das oberste element_node-Element einer DAD-Datei repräsentiert den Wurzelknoten eines XML-Dokumentes. Der zugehörige RDB_node enthält folgende Angaben:

- eine Aufzählung aller mit dem XML-Dokument verbundenen Tabellen,
- die Angabe des Primärschlüssels pro Tabelle, wenn die DAD-Datei zur Dekomposition mit dxxShredXML benutzt wird,
- eine Bedingung, die einen Verbund aller Tabellen über Schlüssel-Fremdschlüssel-Beziehungen erlaubt, und
- Datenbankattribute, anhand derer die XML-Elemente zu ordnen sind.

Für jeden einfachen, textuellen Elementinhalt (#PCDATA) und jedes Attribut eines Elementes ist ein RDB_node für die korrespondierenden text_node- und attribute_node-Elemente anzugeben. Der RDB_node-Knoten gibt im Einzelnen dann notwendige Abbildungsinformationen an:

- den Namen der zugehörigen Relation (Tabelle) und des Attributes (Spalte) in der Datenbank,
- eine optionale Bedingung, die eine Selektion von Attributwerten vornimmt, und
- eine Typangabe in Form eines SQL-Datentyps, wenn die DAD-Datei zur Dekomposition (Shredding) eingesetzt wird, um ein passendes Relationsschema erzeugen zu können.

Die Nutzung der `RDB_node`-Methode zur Dekomposition und Komposition von XML-Dokumenten soll an einer vereinfachten DAD-Datei in Beispiel gezeigt werden.

Beispiel 13-3. Es wird angenommen, dass XML-Dokumente Informationen zu Hotels und Zimmertypen enthalten. Diese Informationen sollen in zwei Datenbanktabellen `hotel` und `zimmertyp` abgelegt werden. Die Angabe der zugehörigen Relationen erfolgt in den `table`-Elementen. Eine Verbundbedingung wird ebenfalls festgelegt (`condition`).

Während der erste `RDB_node`-Knoten die Zuordnung zu Relationsschemata vornimmt, geben die weiteren `RDB_node`-Knoten die Zuordnung von Element- und Attributinhalten zu Datenbankattributen an. Der zweite Knoten bildet den Hotelnamen auf ein entsprechendes Schlüsselattribut der Relation `hotel` ab und der dritte den Zimmerpreis auf die Spalte `Preis`.

```
<DAD>
<Xcollection>
<root_node>
<element_node name="hotel">
  <RDB_node>
    <table name="hotel" key="hname"/>
    <table name="zimmertyp" key="hname, ztyp"/>
    <condition>hotel.hname = zimmertyp.hname</condition>
  </RDB_node>
  <attribute_node name="hname">
    <RDB_node>
      <table name="hotel"/>
      <column name="hname" type="varchar(32)"/>
    </RDB_node>
  </attribute_node>
  <element_node name="zimmertyp">
   ...
    <element_node name="preis">
      <text_node>
        <RDB_node>
          <table name="zimmertyp"/>
          <column name="preis" type="decimal(8,2)"/>
          <condition>preis &lt; 100</condition>
        </RDB_node>
```

```
        </text_node>
      </element_node>
    ...
  </element_node>
</element_node>
</root_name>
</Xcollection>
</DAD>
```

Neben der Angabe des SQL-Typs, einer Festkommazahl mit zwei Nachkommastellen (`decimal(8,2)`), wird eine Bedingung formuliert, die Werte auf Preise unter 100 Euro einschränkt[1] . □

Während die Typ- und Primärschlüsselangaben nur für die Zerlegung mit der `dxxShredXML`-Funktion benötigt werden, um Relationsschemata mit `CREATE TABLE` zu erzeugen, werden alle anderen Angaben sowohl bei der Abbildung auf Datenbankinhalte als auch beim Export als XML-Dokument mit `dxxGenXML` verwandt.

Dynamische Erzeugung von DADs

Die dynamische Erzeugung von DADs erlaubt es, den Abbildungsprozess zu flexibilisieren. Über eine spezielle Methode des XML-Extenders können sowohl Anteile der SQL-Anweisungen (`SQL_stmt`) als auch der XML-Elementkonstruktion und -dekomposition (`RDB_node`) in einer DAD-Datei zur Laufzeit ersetzt werden:

- Mit SQL_OVERRIDE kann der Inhalt des `SQL_stmt`-Elementes einer DAD-Datei und damit die zu verwendende `select`-Anweisung vollständig ersetzt werden.
- Durch XML_OVERRIDE können Bedingungen an Elemente und Attribute angegeben werden. Die Selektionsbedingungen verwenden eine SQL-Syntax und ersetzen die Bedingungen des `RDB_node`-Knotens einer DAD-Datei.

Diese Parameter sind zusammen mit entsprechenden Selektionsbedingungen oder SQL-Anweisungen beim Aufruf einer speziellen Datenbankprozedur (Stored Procedure `dxxRetrieve`) des XML-Extenders, die zur dynamischen Konstruktion und Dekomposition von XML-Dokumenten dient, anzugeben.

13.2.4 TextExtender und Volltextsuche

Völlig orthogonal zur XML-Speicherung mit dem XML-Extender kann der DB2 TextExtender zur Volltext-Indexierung und für das Information Retrieval eingesetzt werden.

Im Zusammenhang mit XML-Dokumenten erlaubt der TextExtender eine strukturierte Volltextsuche basierend auf Abschnitten. Diese Abschnitte sind durch Pfadausdrücke beschreibbar. Intern findet für die Indexierung ein XML-

1. Das Zeichen »<« muss hier aufgrund der Mehrdeutigkeit als Entity spezifiziert werden.

sensitiver Volltextindex Anwendung. Die Nutzung des TextExtender ist unabhängig vom XML-Extender und nicht zwingend notwendig.

Beispiel 13-4. Zur Extraktion aller Beschreibungen von Hotels, in denen das Wort »Seeblick« vorkommt, ist die `contains`-Funktion auf den unter `dscrHandle` abgelegten Volltext in der Relation `hotel` anzuwenden.

```
select beschreibung
from hotel
where contains(dscrHandle,
    'MODEL order SECTION(/beschreibung/feature) "Seeblick"') = 1
```

Die `contains`-Funktion hat in diesem Fall zwei Argumente. Das erste stellt einen Identifikator (Locator Handle) für den Volltext dar, der durchaus sehr groß werden kann und darum speziell verwaltet wird (Locator-Mechanismus). Das zweite Argument in Form einer Zeichenkette spezifiziert die Information-Retrieval-Anfrage, bestehend aus dem Retrievalmodell, dem zu untersuchenden Abschnitt `/beschreibung/feature` und dem Stichwort »`Seeblick`«. □

13.2.5 Weitere Komponenten

Verbunden mit dem XML-Extender sind eine Reihe weiterer Funktionen. Ein DTD Repository kann direkt in DB2 unterhalten werden. DTDs in diesem Repository werden über einen Identifikator DTDID referenziert. So kann durch die Angabe der DTDID in der DAD-Datei eine Validierung gegen eine DTD oder ein Schema beim Import von XML-Dokumenten verlangt werden. Außerdem ist die Angabe allgemeiner Integritätsbedingungen für einzulesende oder zu erzeugende XML-Dokumente über die DAD möglich.

Ein GUI-basiertes Administrationswerkzeug hilft beim Einrichten der XML-Unterstützung für Datenbanken, Tabellen, Spalten und Kollektionen. Es liefert weiterhin Unterstützung beim Erzeugen von DAD-Dateien.

Die Konsistenz der erstellten DAD-Dateien kann mit einem speziellen Werkzeug, dem DAD-Checker, überprüft werden. Neben der Untersuchung der syntaktischen Korrektheit erfolgt eine Validierung von DAD-Dateien, die u. a. fehlende Typangaben, mehrfaches Auftreten von Bezeichnern für Attribut- und Elementknoten und Fehler bei der Abbildung von Tabelleninhalten aufdeckt. Der DAD-Checker ist ein Kommandozeilenwerkzeug und benötigt nicht zwingend eine Datenbankverbindung.

Die Programmierung von XML-Extender-Anwendungen kann wie generell alle DB2-Anwendungen mit Embedded SQL, ODBC, JDBC und SQLJ erfolgen. Speziell für Web-Anwendungen ist eine Integration des XML-Extenders mit *DB2 Net.Data* und der Web-Server-Umgebung *WebSphere* vorgesehen. Net.Data ist eine Skriptsprache zur Generierung dynamischer Webseiten aus *DB2* oder anderen ODBC-konformen Datenquellen. Über einen Makromechanismus kann in

Net.Data-Skripten das Ergebnis von SQL-Anfragen mit nutzerdefinierten Tags versehen und als XML-Dokument ausgegeben werden.

Das System XPERANTO [CFI+00] von Carey, Florescu et al. realisiert die Ausgabe von Inhalten objektrelationaler Datenbanken mit XML-Syntax auf die folgende Weise: Zu existierenden Datenbanken wird eine XML-Sicht gebildet, die die Datenbankinformationen durch eine Schachtelung von Elementen darstellt. An die XML-Sicht werden XML-QL-Anfragen gestellt, die dadurch Datenbankausschnitte auswählen sowie durch die Spezifikation in der `CONSTRUCT`-Klausel die Syntax der Ausgabe definieren. Eine Umsetzung der Grundidee von XPERANTO – externe XML-Sichten auf objektrelationalen Datenbanken – wird in die nächste DB2-Version einfließen. Als Sprache zur Abfrage der Sichten wird jedoch XQuery und nicht XML-QL Anwendung finden.

13.2.6 Einsatzmöglichkeiten

Mit dem XML- und TextExtender bietet DB2 zahlreiche Speicherungsmöglichkeiten für XML an. Mit dem TextExtender können Volltexte u.a. im XML-Dokumentmodell gespeichert werden. Der Zugriff erfolgt über SQL/MM-Fulltext-Erweiterungen. Information-Retrieval-Operationen können durch XPath-Angaben auf Teildokumente eingeschränkt werden. Durch den TextExtender erfolgt eine XML-sensitive Indexierung.

Der XML-Extender stellt einen XML-Datentyp mit XPath-Datenmodell und -methoden zur semistrukturierten Speicherung bereit. Ausgewählte Anteile können für strukturierte Zugriffe indexiert werden.

Mit XML Collections und DAD-Dateien kann eine Abbildung von XML-Dokumenten auf objektrelationale Strukturen und umgekehrt erfolgen. SQL-Anfrageergebnisse, Sichten und komplette Basisrelationen können als XML-Dokument exportiert werden.

13.3 Oracle 9i

Zum Datenbanksystem Oracle 9i wird ein Oracle XML Developer's Kit (Oracle XDK) bereitgestellt. Es enthält Grundbausteine zum Lesen, Manipulieren, Transformieren, Anzeigen und Generieren von XML-Dokumenten. Im Folgenden werden einzelne Funktionen dazu detaillierter dargestellt. Die XML SQL Utility (XSU) umfasst die folgenden drei wesentlichen Komponenten.

- Ausgabe von Datenbankinhalten als XML-Dokumente (XML-Export),
- native Speicherung von XML als XMLType und Abfragen mit XPath sowie
- strukturierte Abbildung von Inhalten aus XML-Dokumenten in Relationen und Attributen.

In den nächsten Abschnitten werden diese Varianten zur Unterstützung der XML-Verarbeitung detaillierter dargestellt.

13.3.1 Export von Datenbankinhalten mit XML-Syntax

Es ist möglich, Daten, die in Oracle-Datenbanken gespeichert sind, als XML-Dokumente auszugeben und damit für andere Anwendungen bereitzustellen. Dabei können die Informationen aus einer Tabelle oder aus einer Sicht ausgegeben werden. Das Ergebnis können sowohl XML-Dokumente als flacher Text sein, oder es können DOM-Bäume für andere Applikationen bereitgestellt werden.

Bevor die genaue Syntax eines solchen Exportes dargestellt wird, sollen zunächst Abbildungsregeln angegeben werden, denen die Bildung der XML-Dokumente folgt.

- Spalten werden auf Elemente der ersten Ebene (top level) abgebildet.
- Einfache Typen (skalare Werte) werden als Element mit `#PCDATA` dargestellt.
- Strukturierte Typen und ihre Attribute werden auf Elemente mit Subelementen für die Attribute abgebildet. Damit werden komplexe Attribute als hierarchisch geschachtelte Elemente dargestellt.
- Aus Kollektionstypen werden Listen von Elementen.
- Objektreferenzen und referentielle Integritätsbedingungen werden auf ID/IDREF innerhalb eines Dokumentes abgebildet.

Zur Illustration der Abbildung soll ein etwas umfangreicheres Beispiel herangezogen werden. Die nachfolgend erzeugten Tabellen werden dabei verwendet.

Beispiel 13-5.

```
CREATE TYPE Adresse_Type AS OBJECT
(    ort     VARCHAR2(30),
     plz     NUMBER,
     strasse VARCHAR2(40),
     nummer  CHAR(4)
);

CREATE TYPE preise_Type AS OBJECT
(   zimmertyp VARCHAR2(20),
    waehrung VARCHAR2(2),
    betrag REAL
);

CREATE TYPE preiseListType AS TABLE OF preise_Type;

CREATE TABLE hotel
(   hotel_name VARCHAR2(20),
    hotel_adr  Adresse_Type,
    preise preiseListType
) NESTED TABLE preise STORE AS nested_preise;
```

Für die Anfrage

```
select * from hotel
```

wird durch die XML SQL Utility XSU das folgende XML-Dokument generiert.

```
<?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <hotel_name>Strand Hotel Huebner</hotel_name>
    <hotel_adr>
      <ort>Rostock-Warnemuende</ort>
      <plz>18119</plz>
      <strasse>Strandstr.</strasse>
      <nummer>12</nummer>
    </hotel_adr>
    <preise>
      <preise_ITEM num="1">
        <zimmertyp>einzelzimmer</zimmertyp>
        <waehrung>EUR</waehrung>
        <betrag>99</betrag>
      </preise_ITEM>
      <!-- weitere Preise -->
    </preise>
  </ROW>
  <!-- weitere Hotels ... -->
  <ROW num="2">
   ...
</ROWSET>
```

□

Mit dieser Vorgehensweise ist die Ausgabe von Datenbankinhalten als XML-Dokumente realisierbar. Die Ergebnisse von SQL-Anfragen (also auch die Inhalte von Relationen oder Views) können als XML-Dokumente ausgegeben werden. Die Syntax dabei ist vorgegeben und im oberen Beispiel dargestellt. Eine individuelle Anpassung der resultierenden XML-Dokumente ist insofern möglich, dass die Tags `<ROWSET>` und `<ROW>` sowie das Attribut `num` weggelassen oder verändert werden können. Die anderen Bestandteile lassen sich durch die SQL-Anfrage beeinflussen, hier kann man zum Beispiel durch Selektion und Projektion beeinflussen, welche Informationen in das XML-Dokument aufgenommen werden sowie durch Umbenennung von Attribute die Tagnamen verändern. Reichen diese Veränderungsmöglichkeiten des Formats nicht aus, so können nachträglich Änderungen des XML-Dokuments durch Einsatz von XSLT realisiert werden. Ein XSLT-Prozessor wird ebenfalls von Oracle bereitgestellt.

13.3.2 XML-Speicherung mit XMLType

Ab der Version Oracle*9i* steht ein neuer Server-Datentyp `XMLType` bereit, der die Speicherung von XML-Dokumenten oder -fragmenten realisiert und Anfragen auf den so gespeicherten Daten ermöglicht. Das Erzeugen einer Instanz vom Typ `XMLType` erfolgt mit `Sys.XMLType.createXML('xml_as_string')`. Intern basiert die Speicherung auf `CLOBs`. Bei Anfragen an die gespeicherten XML-Dokumente kön-

nen XPath-Funktionalitäten verwendet werden. Im Folgenden sollen Beispiele gegeben werden, wie ein solcher XMLType erzeugt werden kann, wie Werte eingegeben und Anfragen realisiert werden können.

Beispiel 13-6.

```
CREATE TABLE hotels
( name          VARCHAR2(40),
  ort           VARCHAR2(35),
  beschreibung  SYS.XMLTYPE,
  ausstattung   SYS.XMLTYPE,
  preise        SYS.XMLTYPE,
  telefonnr     VARCHAR(30)
);
```

Das Einfügen von Tupeln kann zum Beispiel so aussehen:

```
INSERT into hotels (name, ort, beschreibung, ausstattung,
       preise, telefonnr)
VALUES ('Hotel Neptun', 'Warnemuende',
        sys.XMLType.createXML('In Warnemuende ankommen
        und sich zu Hause fuehlen'),
        sys.XMLType.createXML('<schwimmbad>Original-
        Thalassozentrum. Wir haben fuer Sie frisches
        Ostseewasser in das Hotel geholt! </schwimmbad>'),
        sys.XMLType.createXML('<DZ waehrung="Euro">186</DZ>
        <EZ waehrung="Euro">135</EZ>'),
        '0381/54370');
```

□

Bei Ausführen der INSERT-Anweisung wird getestet, ob die XML-Dokumente wohlgeformt sind. Die Gültigkeit der Dokumente wird nicht überprüft, weil im XML-Typ auch Fragmente eines XML-Dokuments gespeichert werden sollen.

Selektion und Extraktion von XMLType-Instanzen

Für die Realisierung von Anfragen lassen sich XPath-Ausdrücke verwenden. Mit der Funktion `extract()` lassen sie sich spezifizieren. Die Zugriffe können syntaktisch in SQL-Anfragen eingebettet werden. Anfragen an XML-Dokumente können damit wie im nächsten Beispiel angegeben erfolgen.

Beispiel 13-7.

```
SELECT ausstattung.extract
       ('schwimmbad/text()').getStringVal()
        "Swimmingpool"
FROM hotels;

Swimmingpool
----------------
Original-Thalassozentrum. Wir haben fuer Sie
frisches Ostseewasser in das Hotel geholt!
```

Anfragen können auch in Datenbankattributen gespeicherte Informationen und als XMLType gespeicherte XML-Dokumente gemeinsam anfragen:

```
SELECT name, ausstattung.extract
       ('schwimmbad/text()').getStringVal() "Swimmingpool",
       preise.extract
       ('/DZ/text()').getNumberVal()
       "Doppelzimmer"
FROM hotels
WHERE ort='Warnemuende';

Name              Swimmingpool                Doppelzimmer
------------------------------------------------------------
Hotel             Original-Thalassozentrum    186
Neptun            Wir haben fuer Sie
                  frisches Ostseewasser
                   in das Hotel geholt!
```

□

Das Ergebnis ist ein Knoten oder eine Knotenmenge, die durch den XPath-Ausdruck beschrieben wird. Mit den Funktionen `getStringVal()` oder `getNumberVal()` kann man auch skalaren Inhalt als Zeichenkette oder numerischen Inhalt als Ergebnis erhalten. Der `XMLType` ist nicht für Änderungsoperationen (UPDATE) auf Teilen geeignet: Müssen die gespeicherten XML-Dokumente verändert werden, so können sie nur vollständig ausgetauscht werden. Der Export der so gespeicherten Dokumente erfolgt mit den gleichen Techniken wie »normale Tabellenspalten«, beide Varianten können auch kombiniert werden, als Ergebnis entsteht ein XML-Dokument.

13.3.3 Speicherung von XML-Dokumenten mit intermedia Text

Eine weitere Möglichkeit zur Speicherung dokumentzentrierter XML-Dokumente besteht durch Verwendung von intermedia Text. Im Datenbanksystem Oracle kann eine Verarbeitung von Volltext-Datentypen durch die ConText-Option erfolgen. Durch sie werden Information-Retrieval-Anfragen in Form von speziellen SQL-Anweisungen unterstützt. Möglich sind Anfragen, die eine Stammwortdeduktion durchführen oder Wildcards verwenden; außerdem kann eine Ähnlichkeits- und Phrasensuche erfolgen, und Synonyme können ermittelt werden. Über die Context-Option erfolgt ein Ranking der Ergebnisse, ein Relevance Feedback durch den Anwender ist ebenfalls möglich. Anfrageterme können mit AND, OR und NOT verknüpft werden, es ist ebenfalls eine Wichtung der Anfrageterme möglich. Eine Suche nach allen Texten, in denen der Wortstamm »Strand« auftritt, sieht wie im folgenden Beispiel aus. Durch die contains-Anweisung wird ein score-Wert im Bereich 0-99 gebildet, der Rankingwerte der Ergebnisse für die Anfrage beinhaltet.

Beispiel 13-8.

```
select score(1) as score, beschreibung from hotels
where contains(beschreibung, 'stem(Strand)', 1) > 0
order by score desc;
```

□

Mit intermedia Text ist ebenfalls eine XML-Unterstützung gegeben, die bewirkt, dass Volltextanfragen und Anfragen zur Struktur der Dokumente kombiniert werden können. Seit der Version *9i* existiert die Möglichkeit, mit XPath auf die gespeicherten Dokumente zuzugreifen, so dass auch Anfragen an XML-Dokumente gestellt werden können, in denen das Markup ausgewertet wird. Die Speicherungsvariante ist für Anfragen geeignet, die sowohl Information-Retrieval-Anfragen als auch eine Auswertung des Markups einbeziehen.

13.3.4 Strukturierte Speicherung von XML

Neben den Varianten zur Speicherung von dokumentzentrierten XML-Dokumenten gibt es auch die Möglichkeit, Elemente und Attribute aus XML-Dokumenten direkt auf Datenbanktabellen und Spalten abzubilden. Die Variante wird auch als *shredded storage* bezeichnet. Die Abbildung der XML-Dokumente auf Datenbankattribute erfolgt, wenn die Syntax der XML-Dokumente der Syntax entspricht, die beim Export von XML-Dokumenten aus Datenbanken erzeugt wird, also der Syntax, die in Beispiel 13-5 angegeben ist. Der »Import« entspricht also der Umkehrung der Exportfunktion. Liegen die XML-Dokumente nicht in dieser Syntax vor, so muss die Zerlegung und Modifikation der Originaldokumente mit XSLT durch den Anwender spezifiziert werden. Mit dieser Variante der Speicherung ist eine Speicherung von Teilen von XML-Dokumenten sowie von vollständigen XML-Dokumenten möglich. Wie die Abbildung erfolgt, wird durch die Transformationen der XML-Dokumente festgelegt, die vom Benutzer durch Verwendung von XSLT definiert werden müssen.

Seit der Version Oracle*8i* werden im XDK (XML Developer's Kit) Bibliotheken bereitgestellt, die den Anwender bei der Verarbeitung von XML-Dokumenten unterstützen.

Diese Bibliotheken enthalten unter anderem:

- XML-Parser (DOM, SAX) für Java, C, C++, PL/SQL
- XSLT-Prozessoren
- XML-Schema-Prozessoren
- XML Class Generator for Java (generiert Java-Klassen)

Damit stehen Tools bereit, mit denen individuelle Applikationen erstellt werden können.

13.3.5 Kombination der Speicherungsvarianten

Es ist auf Basis von Oracle-Datenbanken auch möglich, verschiedene Speicherungsvarianten zu kombinieren. Damit hat man Unterstützung bei der hybriden Speicherung von XML-Dokumenten. Datenzentrierte Anteile können direkt in Datenbanken gespeichert werden, für dokumentzentrierte Anteile eignet sich eine Speicherung als XMLType. Anfragen können dann auf beiden Speicherungsvarianten durchgeführt werden. Auch beim Export lassen sich die verschiedenen Speicherungsformen wieder zusammenführen, so dass die XML-Dokumente wiederhergestellt werden können.

13.3.6 Einsatz

Damit wurde durch den XMLType von Oracle eine Möglichkeit geschaffen, um dokumentzentrierte XML-Dokumente abspeichern zu können. Die XML-Dokumente bleiben erhalten, können also auch verlustfrei wiederhergestellt werden. Eine Anfrage kann durch XPath erfolgen, diese Ausdrücke werden in SQL integriert. Damit ist es auch möglich, die Informationen aus den XML-Dokumenten und andere in Oracle gespeicherte Informationen (der gleichen oder einer anderen Relation) gemeinsam in Anfragen auszuwerten sowie Update-, Insert- und Delete-Anweisungen durchzuführen, bei denen allerdings die Attribute vom Typ XMLTypes nur als Ganzes eingefügt, verändert oder gelöscht werden können. Eine andere Speicherungsmöglichkeit für XML-Dokumente ist durch den Import in Relationen gegeben, hierbei erfolgt eine strukturierte Speicherung der Dokumente. Erforderliche Umwandlungsschritte sind durch Verwendung von XPath möglich, auf diese Weise wird ein benutzerdefiniertes Mapping ermöglicht.

Hybride Speicherungsvarianten, bei denen Teildokumente als XMLType und andere Dokumentanteile in strukturierter Form gespeichert werden, werden bei Oracle ebenfalls ermöglicht. Ein Export von Daten aus Relationen oder Views in XML-Syntax ist möglich. Erfolgt ein Export für Ergebnisse, in denen Attribute vom Typ `XMLType` sind, so werden die entsprechenden gespeicherten XML-Fragmente in das resultierende XML-Dokument eingebettet. Als Ergebnis entsteht ein XML-Dokument, in dem sowohl Informationen, die strukturiert in der Oracle-Datenbank gespeichert waren als auch als `XMLType` gespeicherte XML-Fragmente zusammengeführt werden.

Weitere Informationen zu Oracle 9*i* findet man auf der Webseite [Orac].

13.4 Microsoft SQL Server 2000 und XML-Unterstützung

Microsoft bietet mit dem SQL Server 2000 (im Weiteren kurz SQL Server genannt) verschiedenste Integrationsmöglichkeiten von XML in Datenbankanwendungen. Im Datenbanksystem wird sowohl der Export als auch der Import von XML-Daten unterstützt. Es ist möglich, XPath-Ausdrücke über XML-Dokumente mit der OpenXML-Funktion in SQL-Anfragen einzubeziehen. Die Ergeb-

nisse solcher Anfragen und die Ergebnisse von SQL-Anfragen können mittels einer speziellen `FOR XML`-Klausel als XML-Dokumente dargestellt werden.

Außerdem können relationale Daten über XML-Sichten abgefragt und über so genannte Updategrams von XML aus manipuliert werden. Die XML-Sichten werden dabei mit XDR definiert, einer speziellen, Microsoft-spezifischen Schemabeschreibungssprache.

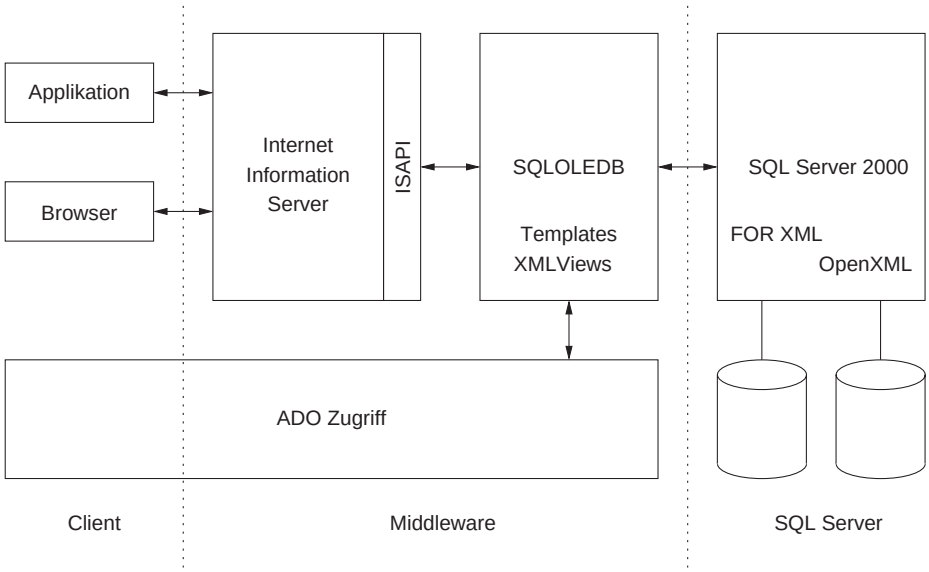


Abb. 13-4: *XML-Unterstützung und SQL Server 2000 (nach [Rys01])*

Bei der Anwendungsentwicklung wird der Nutzer durch eine Integration von XML auf einer Middleware-Ebene unterstützt. Abbildung 13-4 veranschaulicht die XML-Unterstützung des MS SQL Servers. Über einen so genannten SQLOLEDB-Provider wird auf Basis der ActiveX-Kommunikationsplattform der Zugriff auf die XML-Erweiterungen des SQL Server 2000 vom *Internet Information Server* und von der Programmierumgebung ADO aus realisiert.

Die Anbindung in *IIS* erfolgt über die Web-Server-Schnittstelle ISAPI. ADO steht für ActiveX Data Objects und ist eine Programmierschnittstelle zur Arbeit mit verteilten Datenobjekten, die u.a. in einem Datenbanksystem gespeichert sein können.

Die Middleware stellt als zentrale Elemente XML-Sichten und Templates bereit. Ziel ist die Entkopplung von Web- und ADO-Anwendungen vom eigentlichen Datenbanksystem. Über XML-Sichten können relationale Daten mit XML-Techniken angefragt und aktualisiert werden. Templates können dazu enthalten:

- Transact-SQL, Microsofts SQL-Dialekt für die Anwendungsprogrammierung,

- so genannte Updategrams, XML-Dokumente zur Beschreibung von Änderungsoperationen und
- XPath-Anfragen.

XML-Sichten werden etabliert, indem ein XML-Schema mit Abbildungsvorschriften versehen wird. Die Abbildung von XML-Hierarchien kann von und auf relationale Strukturen in Form einer Verbundsicht, genau genommen eines Außenverbundes, erfolgen.

13.4.1 Darstellung von SQL-Anfrageergebnissen als XML

Vom MS SQL-Server 2000 wird eine Erweiterung von SQL um eine `FOR XML`-Klausel vorgenommen. Mit dieser ist es möglich, SQL-Anfrageergebnisse als XML-Dokumente auszugeben. Dabei werden drei Modi unterschieden:

1. `RAW`: erzeugt für jedes Ergebnistupel ein XML-Element ohne Subelemente.
2. `AUTO`: erzeugt ein geschachteltes XML-Dokument mit dem Anfrageergebnis.
3. `EXPLICIT`: erlaubt, die Form und Schachtelung des Ergebnisdokuments in der Anfrage selbst zu bestimmen.

Die `FOR XML`-Klausel wird auf der obersten Ebene wie eine `ORDER BY`-Klausel einer `SELECT-FROM-WHERE`-Anfrage nachgestellt. Sie hat folgende Syntax:

```
FOR XML (RAW | AUTO [, ELEMENTS] | EXPLICIT) [, XMLDATA]
```

Mit `XMLDATA` kann dabei angegeben werden, ob zu dem generierten XML-Dokument auch ein XDR-Schema (XML-Data Reduced) ausgegeben wird.

RAW-Modus

Der erste und einfachste Modus erzeugt ein flaches XML-Dokument zu einer SQL-Anfragen wie folgt. Zu jedem Ergebnistupel wird ein XML-Element mit dem Bezeichner `row` gebildet. Jeder Attributwert des SQL-Ergebnisses wird auf ein Attribut des XML-Elementes abgebildet. Der Spaltenname wird dabei zum Attributnamen. Ist ein Attributwert `NULL`, wird dieses fortgelassen.

Beispiel 13-9. Dieses Beispiel stellt den `RAW`-Modus dar. Dabei wird zu der Anfrage:

```
SELECT Hotel.Name, Zimmertyp.Typ, Zimmertyp.Preis
FROM Hotel, Zimmertyp
WHERE Hotel.Name = Zimmertyp.HName
FOR XML RAW
```

vom MS-SQL-Server dieses XML-Fragment erstellt:

```
<row Name="Hotel Neptun" Typ="EZ" Preis="120"/>
<row Name="Hotel Neptun" Typ="DZ" Preis="170"/>
<row Name="Hotel Neptun" Typ="Suite" Preis="350"/>
```

□

AUTO-Modus

Der AUTO-Modus liefert als Ergebnis eine geschachtelte Elementstrukturen. Dabei wird jede Tabelle der FROM-Klausel als Element repräsentiert. Die Abfolge in der FROM-Klausel bestimmt die Schachtelung der Elemente. Die in der SELECT-Klausel angegebenen Spalten werden im XML-Dokument auf Attribute der zur Tabelle korrespondierenden Elemente abgebildet.

Die Umbenennung von Element- und Attributnamen des XML-Dokumentes ist im begrenzten Rahmen mit SQL-AS-Klauseln möglich. Das kann etwa erforderlich sein, da ein unterschiedlicher Wertevorrat für Bezeichner in SQL und XML existiert.

Beispiel 13-10. Der AUTO-Modus soll am folgenden Beispiel dargestellt werden.

```
SELECT Hotel.Name, Zimmertyp.Typ, Zimmertyp.Preis
FROM Hotel, Zimmertyp
WHERE Hotel.Name = Zimmertyp.HName
FOR XML AUTO
```

erzeugt dieses XML-Dokument:

```
<Hotel Name="Hotel Neptun">
  <Zimmertyp Typ="EZ" Preis="120"/>
  <Zimmertyp Typ="DZ" Preis="170"/>
  <Zimmertyp Typ="Suite" Preis="350"/>
</Hotel>
```

□

Im Gegensatz zum vorhergehenden Beispiel werden nicht der generische Elementname row, sondern der aus dem Tabellennamen abgeleitete eingesetzt.

Man kann in diesem Modus auch angeben, dass im resultierenden XML-Dokument keine Attribute, sondern Subelemente zur Darstellung der Spalten und ihrer Werte eingesetzt werden sollen. Dazu wird das Schlüsselwort ELEMENTS ergänzt.

Beispiel 13-11. In diesem Fall sieht die Anfrage und das Ergebnis aus Beispiel 10 wie folgt aus:

```
SELECT Hotel.Name, Zimmertyp.Typ, Zimmertyp.Preis
FROM Hotel, Zimmertyp
WHERE Hotel.Name = Zimmertyp.HName
FOR XML AUTO, ELEMENTS
```

Generiert wird dieses XML-Dokument:

```
<Hotel>
  <Name>Hotel Neptun</Name>
  <Zimmertyp><Typ>EZ</Typ><Preis>120</Preis></Zimmertyp>
  <Zimmertyp><Typ>DZ</Typ><Preis>170</Preis></Zimmertyp>
  <Zimmertyp><Typ>Suite</Typ><Preis>350</Preis></Zimmertyp>
</Hotel>
```

Die Attribute der Relation Zimmertyp sind nicht als XML-Attribute, sondern vielmehr als Subelemente dargestellt. □

EXPLICIT-Modus

Sowohl der RAW- als auch der AUTO-Modus sind nur in der Lage, Ergebnisse in einem festen Format zu liefern. Der dritte vom SQL Server angebotene EXPLICIT-Modus erlaubt die Ausgabe beliebiger XML-Dokumente. Dabei kann eine beliebige hierarchische Schachtelung erreicht werden, die unabhängig von der spezifizierten Folge von Basisrelationen ist. Es ist dabei möglich, Elemente mit Daten über verschiedene Tabellen zu konstruieren. Über spezielle Schlüsselworte können Schlüssel-Fremdschlüsselbeziehungen auf ID/IDREFS-Attribute abgebildet werden.

Dieser Modus ist jedoch wesentlich aufwändiger für den Anwender, weil die Art der Abbildung genau spezifiziert werden muss.

Die Generierung der XML-Dokumente erfolgt in zwei Schritten:

1. Bearbeitung einer Anfragesicht, und
2. Serialisierungsprozess zur Ergebnisaufbereitung.

Der EXPLICIT-Modus erwartet das Ergebnis der Anfragesicht in einem speziellen Format, dem Universal-Relation-Format. Die Sicht muss zwei spezielle Spalten Tag, Parent sowie weitere Spalten mit kodierten Spaltennamen und einer bestimmten Ordnung im Ergebnis erzeugen. Die Anzahl der Spalten sowie deren Bezeichnungen und Inhalte beeinflussen das gewünschte Ergebnisdokument.

Universal-Relation

Eine Universal-Relation hat die folgenden Spalten:

- Tag stellt die Tag-Nummer und Hierarchieebene des aktuell auszugebenden Elementes dar,
- Parent repräsentiert die Tag-Nummer des Elternknotens. Tupel mit einem Nullwert (NULL oder 0) werden auf der obersten Ebene angeordnet sowie
- weitere Spalten, deren Name Attribut-, Elementnamen, Hierarchieebene und Knotentyp kodieren.

Die Kodierung der Spaltennamen erfolgt nach einem festen Muster:

```
Name! Ebene! Subname[! Type]
```

Durch die Angabe von Name und Ebene erfolgt eine Zuordnung von Elementnamen und Hierarchieebene. Weiterhin wird mit Subname der Bezeichner des Attributes oder Subelementes festgelegt. Folgt eine Typangabe, wird nicht ein Attributknoten (Normalfall), sondern spezielle Knotentypen, etwa IDREFS, CDATA-Abschnitte oder Subelemente erzeugt. Mit der Angabe von ELEMENT wird der Wert des Datenbankattributs als #PCDATA-Inhalt eines Elementes ausgegeben. Weiterhin kann z.B. durch XMLTEXT der Wert eines Datenbankattributes als literales XML interpretiert werden und literal in das Ergebnis übernommen werden.

Der Serialisierungsprozess geht davon aus, dass die Tupel, die einen bestimmten XML-Knoten repräsentieren, unmittelbar von den Tupeln der Kindsknoten gefolgt werden. Dies entspräche einer Depth-first-Traversierung des gewünschten XML-Dokumentbaumes. Erreicht wird dies durch explizit anzugebende `ORDER BY`-Klauseln.

Ob die Sicht, wie im nachfolgenden Beispiel, dynamisch erzeugt wird oder ob etwa eine temporäre Tabelle speziell dafür aufgebaut wird, ist unerheblich. Entscheidend für den `EXPLICIT`-Modus ist der einzuhaltende Tabellenaufbau einer Universal-Relation.

Beispiel 13-12. Zur Erzeugung eines Dokumentes mit Angaben zum Hotel und den angebotenen Zimmerarten wird diese SQL-Anfrage unter Nutzung des `EXPLICIT`-Modus gestellt.

```
SELECT 1 AS Tag,
  NULL AS Parent,
  Hotel.Name AS "Hotel!1!Name",
  NULL AS "Zimmertyp!2!Typ"
FROM Hotels
UNION ALL
SELECT 2, 1, Hotel.Name, Zimmertyp.Typ
WHERE Hotel.Name = Zimmertyp.HName
ORDER BY [Hotel!1!Name], [Zimmertyp!2!Typ] FOR XML EXPLICIT
```

Im ersten Schritt der Bearbeitung entsteht folgende Tabelle:

Tag	Parent	Hotel! 1! Name	Zimmertyp! 2! Typ
1	NULL	Neptun	NULL
2	1	Neptun	EZ
2	1	Neptun	DZ
2	1	Neptun	Suite
1	NULL	Am Leuchtturm	NULL
2	1	Am Leuchtturm	EZ
2	1	Am Leuchtturm	DZ

Durch den Serialisierungsprozess wird daraus dieses XML-Fragment erzeugt:

```
<Hotel Name="Neptun">
  <Zimmertyp Typ="EZ"/>
  <Zimmertyp Typ="DZ"/>
  <Zimmertyp Typ="Suite"/>
</Hotel>
<Hotel Name="Am Leuchtturm">
  <Zimmertyp Typ="EZ"/>
  <Zimmertyp Typ="DZ"/>
</Hotel>
```

Durch die Angabe von z.B. »[Zimmertyp! 2! Typ! element]« wird dabei erreicht, dass der Spalteninhalt von Zimmertyp.Typ als Elementinhalt anstatt eines Attributwertes ausgegeben wird. □

13.4.2 Relationale Sichten auf XML mit OpenXML

Mit der OpenXML-Funktion ist der Zugriff von SQL-DB-Anwendungen aus auf XML-Daten möglich. Die Eingabe ist dabei ein XML-Dokument. Zu diesem wird eine interne Repräsentation mit der Funktion sp_xml_preparedocument erstellt. Der Zugriff auf Dokumentteile des XML-Dokumentes erfolgt mit der Funktion OpenXML. Die interne Repräsentation kann mit sp_xml_removedocument wieder freigegeben werden.

OpenXML ist ein so genannter *Rowset Provider*, eine Funktion, die eine Tupelmenge in Tabellenform liefert. Diese Funktionen liefern nicht einen atomaren Funktionswert sondern eine Menge von Tupeln. Diese können in SQL-Ausdrücken überall dort stehen, wo syntaktisch eine Relation erwartet wird, z.B. in der FROM-Klausel.

Die Funktion OpenXML hat folgende Parameter:

```
OpenXML(Docid int [in],
  Pfadausdruck nvarchar [in],
  Flags byte [in])
WITH (Schemadeklaration | Tabellenname)]
```

Das Schema der Ergebnisrelation von OpenXML kann zwei Formen haben:

1. so genannte *Edge Table*, die eine Graphdarstellung der XML-Knotenhierarchie ist. Sie beinhaltet unter anderem Informationen über die Knoten und die jeweiligen Vorgängerknoten sowie
2. *Shredded Rowset*, das eine gezielte Extraktion von Elementen oder Attributen aus dem Dokument erlaubt. Das Ergebnisschema wird über Flags und die WITH-Klausel gesteuert.

Weiterhin ist Docid ein Verweis (*engl.* Handle) auf die interne Repräsentation des Dokumentes. Er wird von sp_xml_preparedocument als Funktionswert geliefert.

Die anderen Parameter von OpenXML haben folgende Bedeutung:

- Pfadausdruck ist ein XPath-Ausdruck, der das zu extrahierende Dokumentfragment spezifiziert. XPath-Ausdrücke sind eine XPath-1.0-Teilmenge.
- Mit der WITH-Klausel kann eine Schemadeklaration wie in der SQL-CREATE TABLE-Klausel erfolgen. Spaltennamen in Schemadeklaration verweisen auf Attribut- oder Elementnamen der durch den Pfadausdruck extrahierten Knotenmenge. Weiterhin gibt es eine Angabe des zu verwendenden SQL-Datentyps und evtl. ein Verweis auf den zu benutzenden XML-Knoten in XPath-Syntax.
- Durch Flags wird die Abbildung von Attribut- (Flags = 1) oder Elementnamen (Flags = 2) auf Spaltennamen gesteuert.

Edge Table-Format

Tabelle 13-1 zeigt das so genannte Edge Table-Format. Wenn `OpenXML` im Ergebnis dieses Format liefert, erfolgt in der Regel eine Weiterverarbeitung in der Anwendung direkt oder mit geschachtelten SQL-Anfragen.

Spaltenname	Bedeutung
`id`	bezeichnet eindeutig einen Knoten eines Dokumentes, die Wurzel hat die `id`.
`parentid`	verweist auf den Elternknoten bei einem Element, bei Textknoten z. B. auf das zugehörige Attribut oder Element.
`nodetype`	identifiziert den Knotentyp unter Verwendung des DOM-Nummerierungschemas für Knotenarten.
`localname`	enthält den Element- oder Attributnamen.
`prefix`	stellt den Namensraumpräfix dar.
`Namespaceuri`	ist, falls vorhanden, die URI des zugehörigen Namensraumes.
`datatype`	gibt den Datentyp von Elementen und Attributen an. Er wird aus der DTD oder dem angegebenen Schema abgeleitet.
`prev`	enthält die Knotennummer des vorhergehenden Geschwisterknotens.
`text`	nimmt den Attributwert oder Elementinhalt auf.

Tab. 13-1: *Format der Edge Table*

Beispiel 13-13. Alle `Zimmertyp`-Elemente des durch »`@doc`« referenzierten Dokumentes im Edge-Table-Format liefert diese SQL-Anfrage:

```
SELECT * FROM OpenXML(@doc, '/ROOT/Hotel/Zimmertyp')
```

Bei der interaktiven Verwendung wird eine Tabelle mit den in Tabelle 13-1 angegebenen Spalten ausgegeben. □

Shredded Rowset

Bei diesem Format können zwei Abbildungen erzeugt werden:

- eine attributzentrierte Abbildung (`Flag = 1`) von Attributen auf Datenbankattribute sowie
- eine elementzentrierte Abbildung (`Flag = 2`) von Elementen verschiedener Hierarchieebenen auf Datenbankattribute.

Beispiel 13-14. Attributzentriert abgebildet wird etwa mit

```
SELECT *
FROM OpenXML(@doc, '/ROOT/Hotel', 1)
  WITH (
    Name varchar(10),
    Sterne integer
  )
```

Eine elementzentrierte Abbildung über verschiedene Hierarchieebenen wird mit dieser Anweisung erreicht:

```
SELECT *
FROM OpenXML(@doc, '/ROOT/Hotel/Zimmertyp', 2)
  WITH (
    Name varchar(20) '../Name',
    Type varchar(10) 'typ',
    Preis integer '@preis'
  )
```

Ergebnis ist eine flache Relation, deren Werte aus unterschiedlichen Elementen `/Hotel/Name`, `/Hotel/Zimmertyp/Typ` und einem Attribut `/Hotel/Zimmertyp/@preis` des XML-Dokuments stammen. □

13.4.3 XML-Sichten auf relationale Daten

Der SQL Server bietet die Möglichkeit, XML als logisches Modell zu verwenden und auf diesem Anfragen und Änderungsoperationen vorzunehmen. Die physische Speicherung erfolgt im relationalen Modell des SQL Servers.

Die Abbildung zwischen der XML-Sicht auf logischer Ebene und der physischen Speicherung in der Datenbank wird durch ein annotiertes Schema mit XDR (XML-Data Reduced) beschrieben. Für die Schemadeklarationen selbst wird auch XML und üblicherweise der Namensraum `xml-data` benutzt.

Die Annotationen referenzieren das relationale Schema, in der Regel über den `xml-sql` Namensraum. Die Verarbeitung der Sichten erfolgt durch XML-Tools, etwa in IIS und ADO. Intern erfolgt eine Abbildung auf SQL, die mit XSLT implementiert ist.

Sichtendefinition mit XDR

Die Beschreibung der XML-Sichten erfolgt mit XML-Data Reduced (XDR), einer speziellen Schemabeschreibungssprache von Microsoft. Sie besitzt selbst eine XML-Syntax und unterstützt Namensräume. Es werden eine Vielzahl an Grunddatentypen geboten, die sprachlichen Möglichkeiten gehen ansonsten wenig über DTDs hinaus.

Beispiel 13-15. Es folgt eine einfache Schemadefinition in XDR:

```
<AttributeType name='hname' dt:type='string'/>
<AttributeType name='ztyp' dt:type='enumeration'
   dt:values='EZ DZ Suite'/>
<ElementType name='zimmertyp'>
   <attribute type='ztyp'/>
   <attribute type='hname'/>
</ElementType>
```

und ein Beispiel, wie die Abbildung auf ein relationales Schema erfolgen kann:

```
<schema xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:sql="urn:schemas-microsoft-com:xml-sql">
<ElementType name="zimmertyp" sql:relation="Zimmertyp">
  <attribute type="ztyp" sql:field="ZTyp"/>
  <attribute type="hname" sql:field="HName"/>
</ElementType>
</schema>
```

Die beiden Namensräume referenzieren zum einen die Grunddatentypen von XDR mit `dt` und zum anderen die Erweiterungen für die Annotationen zur Abbildung auf die relationalen Daten mit `sql`. □

Abfragen sind z. Z. mit XPath, zukünftig auch mit XQuery möglich. Neben diesen Anfragemöglichkeiten können auch Änderungen an den XML-Sichten vorgenommen werden, die an die relationale Datenbank propagiert werden. Dies erfolgt mit so genannten Updategrams.

Laden von Massendaten (Bulk Load)

Alternativ zu OpenXML, mit dem XML-Dokumente zerlegt und per `INSERT` in die Datenbank eingefügt werden können, erlaubt ein spezielles Objekt `SQLXMLBulkLoad` das Laden von großen Mengen an XML-Daten in die Datenbank (so genanntes Bulk Load). Über Eigenschaften dieses Objektes kann der Ladevorgang gesteuert werden, z. B. wird über ein annotiertes XDR-Schema der Abbildungsvorgang gesteuert. Weitere Eigenschaften beeinflussen das Transaktionsverhalten oder geben Integritätsbedingungen, die beim Ladevorgang einzuhalten sind, an.

Auch wenn der Ladevorgang über XML-Sichten definiert wird, läuft er direkt im Datenbank-Server ab und nicht in der Middleware.

13.4.4 Zusammenfassung der Möglichkeiten

Der MS SQL Server 2000 bietet drei Möglichkeiten der Integration von XML in Datenbankanwendungen. Zwei im Datenbanksystem angelagerte Techniken erlauben es, von SQL aus auf XML-Inhalte zuzugreifen und dabei eingeschränkte Pfadausdrücke zu verwenden (`OpenXML`) sowie Anfrageergebnisse als XML zu präsentieren (`WITH XML`-Klausel).

Die dritte Variante ist ein Middleware-Ansatz und verwendet XML als logisches Modell. Über annotierte XDR-Sichten werden die relational gespeicherten Daten als XML dargestellt. Sowohl der Zugriff über XPath und zukünftig XQuery als auch die Manipulation mit Updategrams erfolgt über das XML-Dokumentenmodell.

13.5 eXtensible Information Server (XIS) von eXcelon

Die Firma eXcelon bietet zur Verwaltung und Speicherung von XML-Dokumenten den eXtensible Information Server an, der eine ganze Sammlung von Tools be-

reitstellt. Der eXtensible Information Server wird verwendet, um Daten und Dokumente in verschiedenen Formaten zu integrieren, in andere Formate zu transformieren und für verschiedene Anwendungen bereitzustellen.

Abbildung 13-5 zeigt die Architektur des eXtensible Information Servers.

Die Kernkomponente des eXtensible Information Server realisiert die Speicherung von XML-Dokumenten auf der Basis des *objektorientierten Datenbanksystems ObjectStore*. Dabei werden die XML-Dokumente vor der Abspeicherung in die Datenbank geparst und bezüglich der Gültigkeit gegenüber einer DTD oder einem XML-Schema überprüft.

eXcelon realisiert eine native Speicherung von XML-Dokumenten auf der Basis der Klassenstruktur des *Document Object Model* (DOM). Das heißt, XML-Dokumente werden vollständig gespeichert, es kann auf alle Teile der Dokumente zugegriffen werden und die Dokumente können verlustfrei wiederhergestellt werden. Die DOM-Struktur der XML-Dokumente wird in eine objektorientierte Datenbank abgelegt.

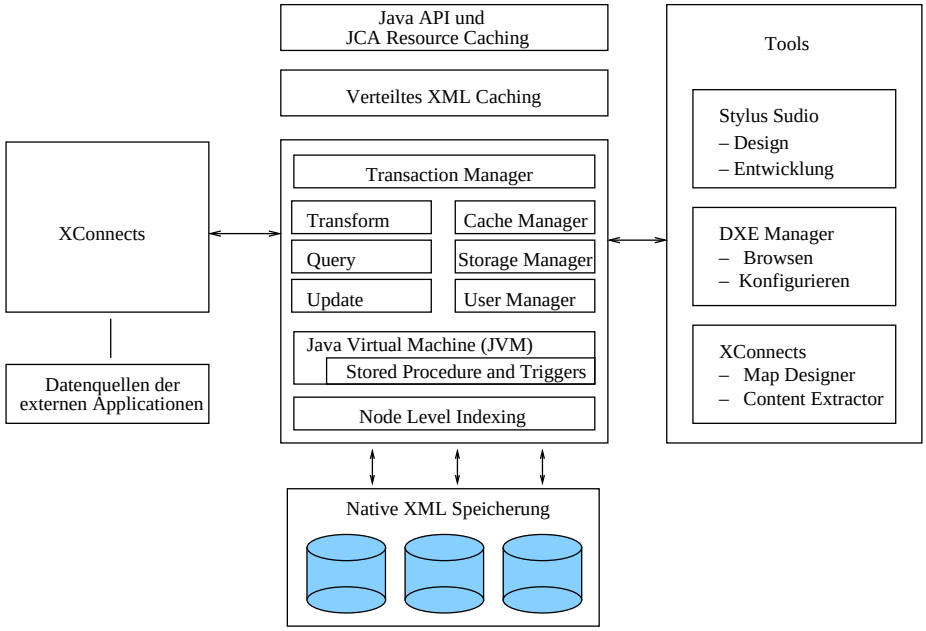


Abb. 13-5: *Architektur des eXcelon eXtensible Information Server*

Dazu wird eine Datenbank eingesetzt, deren Struktur durch die Klassenstruktur des Document Object Model definiert wird. Die Struktur der Datenbank ist damit immer konstant und unabhängig vom Aussehen der konkreten XML-Dokumentkollektionen. Durch diese Speicherungsmethode lassen sich demzufolge Dokumente mit einer DTD, mit einem XML-Schema oder auch ohne eine explizit

dargestellte Struktur speichern. Das ist speziell für die Integration von Daten aus einer Vielzahl dynamischer Quellen sehr hilfreich.

Einzelne Elemente oder Attribute, die auf diese Weise gespeichert werden, können zusätzlich mit einem *Index* versehen werden, der den Zugriff auf die Dokumente beschleunigt. Der Index ist so aufgebaut, dass für die zu indexierenden Elemente oder Attribute ein direkter Pointer in die Datenbank mit den gespeicherten Informationen existiert. Es ist die Anlage von Struktur-, Text- und Werteindexen möglich. Weiterhin wird die Technologie von Verity eingesetzt, um Text-Retrieval in beliebigen Quellformaten zu unterstützen.

Als *Anfragesprache* zum Zugriff auf die gespeicherten XML-Dokumente wird XPath verwendet. Die Angabe von XPath-Ausdrücken ist dabei direkt möglich. Alternativ dazu existiert ein Tool, das eine grafische Unterstützung bei der Generierung von XPath-Anfragen bietet. Darüber hinaus ist zum Zeitpunkt des Erscheinens des Buches bereits ein initialer XQuery-Support implementiert und im System vorhanden. XQuery ist gegenwärtig noch nicht als Empfehlung vom W3C verabschiedet worden.

Geparste und in der XML-Datenbank abgelegte Dokumente können verändert werden; das wird ebenfalls über die Benutzerschnittstelle unterstützt und in eine von eXcelon entwickelte *Update-Sprache*, die sich an SQL und XPath anlehnt, übersetzt. Die Update-Operationen werden auf der Datenbank, die die DOM-Informationen speichert, ausgeführt. Die interne Speicherungsstruktur speichert die einzelnen Nodes eines XML-Dokumentes. Bei Updates müssen nur die Inhalte der Nodes eines XML-Dokumentes geändert werden, die durch das Update verändert werden.

Auf den eXtensible Information Server kann durch ein API für Java zugegriffen werden.

Der eXtensible Information Server erlaubt den Im- und Export von über 80 verschiedenen externen Datenquellen. Dabei können nicht nur Daten aus anderen Datenbanken (wie Oracle und Paradox) importiert werden, sondern es können auch andere Datenformate (wie zum Beispiel Excel, Lotus Notes, Messaging-Systeme wie MQSeries) und Anwendungen (wie SAP) integriert werden. Bei der Verwendung von Informationen aus anderen Datenformaten ist es möglich (und notwendig), ein Mapping herzustellen, das die Zuordnung zwischen den Informationen aus den externen Quellen und den gespeicherten XML-Dokumenten herstellt. Im XIS ist dies durch die `XConnects` genannte Komponente möglich.

Ein Export der gespeicherten XML-Dokumente in beliebige Formate ist durch den Einsatz von XSLT möglich. Dabei kann vom Benutzer individuell bestimmt werden, in welcher Weise die Abbildung erfolgen soll.

eXcelon hat noch eine Reihe weiterer Produkte in den XIS integriert, wie zum Beispiel einen XML-Editor, einen XML-Schema-Editor, einen XSLT-Editor sowie einen XML-XML-Transformator, der XSLT als Output erzeugt.

Das System eXcelon unterstützt eine native Speicherung von XML-Dokumenten, die auf logischer Ebene auf dem Document Object Model basiert und auf physischer Ebene das objektorientierte Datenbanksystem ObjectStore einsetzt. Eben-

falls in das System integriert sind Tools, die eine konzeptuelle Modellierung von XML-Dokumenten und Schemata ermöglichen sowie eine grafische Unterstützung bei der Spezifikation von Anfragen und Updates bieten.

Genauere Informationen zum eXtensible Information Server von eXcelon findet man auf der Webseite [Exce].

13.6 Tamino

Tamino ist der von der Software AG entwickelte XML-Server. Tamino stellt eines der bekanntesten nativen XML-Datenbanksysteme dar. Im Gegensatz zu den auf objektrelationalen oder objektorientierten Datenbanken aufsetzenden Lösungen wurde es vollständig für XML-Dokumente entwickelt. An allen Schnittstellen des Systems werden XML-Instanzen und -Schemata eingesetzt. Bei der Entwicklung wurden jahrelange Erfahrungen aus der Entwicklung des hierarchischen Datenbanksystems Adabas genutzt.

Tamino hat eine Transaktionsverwaltung und ist für den Mehrnutzerbetrieb konzipiert. In Abbildung 13-6 ist der generelle Aufbau des Systems zu erkennen.

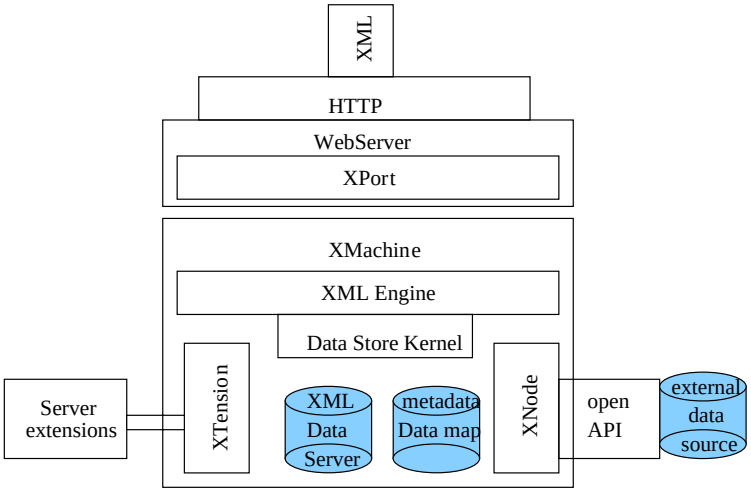


Abb. 13-6: *Architektur des XML-Servers Tamino*

Man sieht, dass der Zugriff auf den XML-Server durch das HTTP-Protokoll erfolgt. Damit werden Zugriffe über das Web ermöglicht. In der URL wird angegeben, welche Datenbank verwendet werden soll und wo das Dokument innerhalb der Datenbank lokalisiert ist; außerdem wird die Anfrage dort selbst spezifiziert.

Die Basisfunktion des Systems ist die Speicherung und das Retrieval von XML-Dokumenten. Abbildung 13-7 stellt dar, wie beide Prozesse mit Tamino erfolgen.

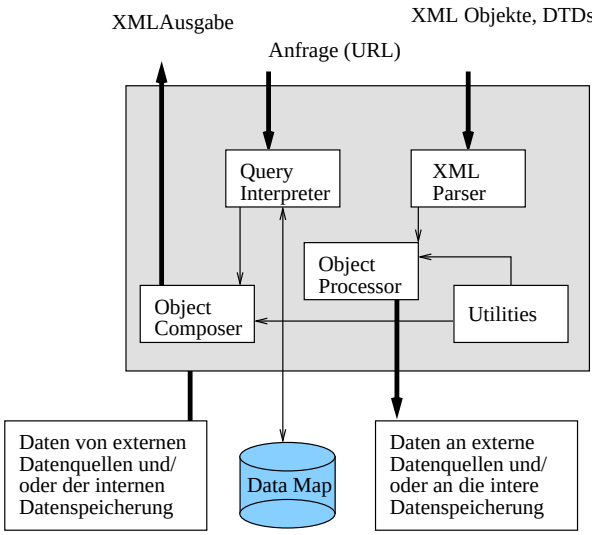


Abb. 13-7: *Speicherung und Retrieval von XML-Dokumenten*

Tamino speichert XML-Dokumente nativ, es erfolgt keine Dokumenttransformation. XML-Prozessoren DOM und SAX stehen dazu innerhalb des Systems bereit. Tamino unterstützt gegenwärtig die Speicherung von XML-Dokumenten mit oder ohne DTD, aus XML Schema werden bisher die vordefinierten Datentypen unterstützt. Weitere Anteile von XML Schema werden in Zukunft schrittweise ergänzt.

Die unterstützte Anfragesprache ist gegenwärtig XPath 1.0 mit speziellen Erweiterungen. Zukünftige Entwicklungen erfolgen in Richtung XQuery 1.0 auf der Basis von XPath 2.0. XQuery ist gegenwärtig noch keine Empfehlung des W3C, befindet sich also noch in der Entwicklung. Das Ergebnis zu einer Anfrage sind entweder die Originaldokumente oder Ausschnitte aus Dokumenten, die in der Anfrage spezifiziert werden. Die Komponente X-Machine enthält einen Parser für die Anfragesprache. Die Anfragen können sowohl an nativ gespeicherte XML-Dokumente als auch an externe Datenquellen wie MS-Office-Dokumente, Audio- und Videodateien oder PDF-Dateien gestellt werden. Die Komponente interagiert mit dem Object-Composer, der die Ergebnisdokumente erstellt. Die Ergebnisse werden ebenfalls aus nativ gespeicherten oder externen Informationen zusammengesetzt.

Die Komponente X-Node stellt Verbindung zu anderen Datenbanken, z.B. relationalen Datenbanken her, kann also den Import oder Export von Daten realisieren. Wie dabei die Zuordnung erfolgt, wird durch Metadaten, die in der Data Map gespeichert sind, definiert. Data Map ist die Wissensbasis des Tamino Servers. Es enthält XML-Metadaten, wie Schemainformationen, Stylesheets, und relationale Schemata. Die Abbildungsvorschriften, die beschreiben, wie XML-Objekte gespeichert und zusammengesetzt werden, sind dort niedergelegt. Auf diese Weise können heterogene verteilte Datenquellen durch Einsatz von Tamino

ausgewertet werden. Das Tamino XML Server Administration Tool bietet eine einfache grafische Schnittstelle für die Festlegung des Mapping an.

Bei der Speicherung der XML-Dokumente ist eine Indexierung möglich. Es werden drei Arten von Indexen angelegt, ein wertbasierter Index, ein textbasierter Index und ein Strukturindex. Der *wertbasierte Index* kann eingesetzt werden, wenn das exakte Vorkommen bestimmter Suchbegriffe im XML-Dokument ermittelt werden soll. Der wertbasierte Index kann auch für andere Datentypen als Zeichenketten eingesetzt werden, zum Beispiel für numerische Datentypen. Der *textbasierte Index* unterstützt demgegenüber den Zugriff auf Texte. Der Strukturindex erlaubt Anfragen über die Struktur des XML-Dokumentes. Eine Anfrageoptimierung erfolgt, dabei werden durch die Anfragen Indexinformationen und vorhandene Schemainformationen ausgenutzt. Die Komponenten XTension erlaubt benutzerdefinierte Funktionen, die Dokumentteile auslesen oder auf Dokumentteile angewendet werden.

Die Vorteile von Tamino kommen besonders bei der Speicherung und beim Retrieval von dokumentzentrierten und semistrukturierten XML-Dokumenten zum Tragen. Durch die Möglichkeit, heterogene externe Datenquellen auszuwerten, eröffnen sich weitere Anwendungsfelder.

Weitere Informationen zum System findet man in der Artikeln [ScWä00] und [Schö01] sowie auf der Webseite [Tami].

13.7 Infonyte-DB

Das System Infonyte verwendet zur Speicherung von XML-Dokumenten eine persistente Implementierung des Document Object Model. Jedes wohlgeformte XML-Dokument kann ohne weitere Schema- oder DTD-Informationen verarbeitet werden.

Das System enthält einen Web-fähigen Anfrageprozessor für die gegenwärtig gängigen XML-Anfragesprachen XQL und XPath 1.0. Eine weitere Schnittstelle ist das DOM Application Programmers Interface. Transformationen der XML-Dokumente werden durch die Integration von Apache Xalan ermöglicht.

Infonyte-DB unterstützt damit die direkte, native Speicherung von XML und einen einfachen Zugriff auf die gespeicherten Informationen. Die Infonyte-DB bietet für XML-Anwendungen eine skalierbare und transparente Komponente zur Speicherung und Abfrage von XML-Daten.

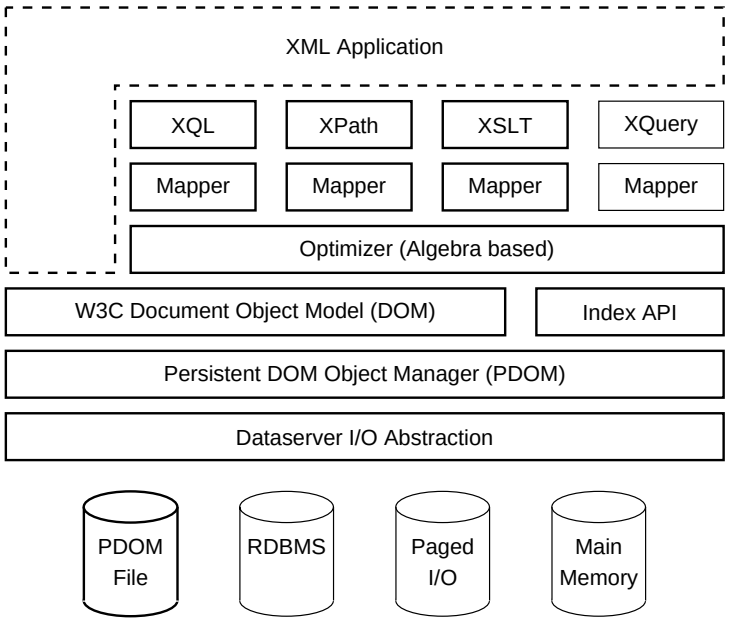


Abb. 13-8: *Architektur der Infonyte-DB*

Das System baut nicht auf existierende Datenbanken auf, sondern entwickelt eigene Komponenten zur physischen Speicherung, die an die Strukturen von XML-Dokumenten optimal angepasst sind. Bei der Speicherung erfolgt eine automatische und für den Anwender transparente Indexierung. XML-Anfragen werden algebraisch optimiert. Weiterhin wurde eine selbstoptimierende Cache-Strategie für die Speicherung von DOM-Informationen entwickelt.

Durch die komplette Neuentwicklung für die Speicherung und Verwaltung von XML-Dokumenten handelt es sich bei der Infonyte-DB um eine schlanke und gleichzeitig leistungsfähige Implementierung.

13.8 POET Object Server Suite

Zum objektorientierten Datenbanksystem POET wird in Form einer Middleware eine Komponente angeboten, die der Speicherung von XML-Dokumenten in POET-Datenbanken dient. Dazu wird folgendes Vorgehen unterstützt:

- Aus einer XML-Schema-Beschreibung werden spezielle Java-Klassen generiert.
- In einem zweiten Schritt wird für diese ein zugehöriger Datenbankentwurf erzeugt.

Komponenten, mit denen Java-Klassen in POET-Datenbanken gespeichert werden, sind schon seit längerer Zeit verfügbar. Ziel dabei ist es, dem Entwickler den

Zugriff auf das XML-Dokument über spezielle Methoden der Java-Klassen, die `marshal` und `unmarshal` heißen, bereitzustellen.

Alternativ können Anfragen an die objektorientierte Datenbank über OQL gestellt werden. Geeignet ist das Verfahren für datenzentrierte XML-Dokumente, die dann strukturiert in Datenbanken gespeichert werden und über Datenbankanfragen ausgewertet werden sollen.

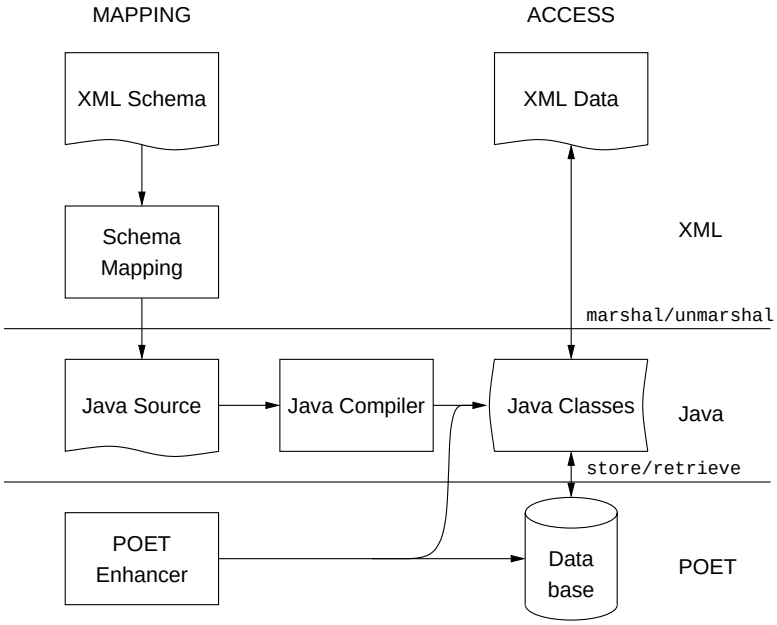


Abb. 13-9: *POET: XML-Speicherung über Java-Klassen*

Die Abbildung 13-9 zeigt im Überblick die Speicherung von XML in POET über Java-Klasssen. Dabei sind die Ebene der Abbildung (Mapping) und des Zugriffes (Access) zu unterscheiden. Der Abbildungsvorgang stellt einen zweistufigen Prozess dar:

1. Ausgehend von einem XML-Schema werden Java-Klassen mittels eines Abbildungswerkzeuges erzeugt. Das XML-Schema wird auf ein objektorientiertes Schema abgebildet.
2. Die generierten Java-Klassen werden persistent in einer POET-Datenbank gespeichert.

Zur Abbildung des XML-Schemas wird das freie Werkzeug *Castor*[2] benutzt. Die verwendeten Regeln sind sehr ähnlich zu der von Bourret beschriebenen Technik (siehe Kapitel 5.7). Um Java-Klassen persistent zu speichern, werden die Java-

2. `http://castor.exolab.org/`

Klassendateien mit einem speziellen Werkzeug (dem so genannten Enhancer) auf Bytecode-Ebene erweitert. Die so modifizierten Klassen sind in der Lage, Instanzen explizit über Erreichbarkeit dauerhaft (persistent) in der POET-Datenbank abzulegen.

Der Zugriff erfolgt auf der Ebene der in Java zu erstellenden Anwendung. Die generierten Java-Klassen haben Methoden, um Instanzen sowohl persistent zu speichern (`store`, `retrieve`) als auch die Instanzen mit Werten aus XML-Dokumenten zu belegen oder sie als XML darzustellen (`marshal`, `unmarshal`). Für Anfragen steht neben dem ODMG Java Binding die Anfragesprache OQL zur Verfügung. Mit der neuen Produktgruppe *FastObjects* von POET kann ebenfalls der JDO- Standard (Java Data Objects) für persistente Java-Anwendungen eingesetzt werden.

Alternativ zur schemabasierten Abbildung gibt es einen Zugang, der DOM-Strukturen in einer POET-Datenbank persistent speichert. Diese Variante ist notwendig, wenn XML-Dokumente gespeichert werden sollen, zu denen kein Schema vorhanden ist, oder wenn das Schema sich häufig ändert. Dabei wird ein feststehendes Datenbankschema erstellt, das die Klassenstruktur des Document Object Model abbildet. In die Datenbank werden die Informationen aus dem XML-Dokument gespeichert. Anfragen können auch hier durch Einsatz von OQL gestellt werden. Diese Variante findet u.a. in einigen kommerziellen Content-Management-Systemen, die POET als Basis zur Datenverwaltung einsetzen, Anwendung.

13.9 Zusammenfassung und Ausblick

In diesem Kapitel wurde eine Vielzahl von Systemen vorgestellt, mit denen XML-Dokumente gespeichert und die Informationen angefragt werden können.

Die Darstellungen der Eigenschaften und Merkmale der XML-Server ist im Kapitel weitaus kürzer als die Darstellungen zu den objektrelationalen Datenbanken. Das hat keinen wertenden Charakter, sondern liegt daran, dass hier stärkere Bezüge auf die vorherigen Kapitel des Buches bestehen. Zum Beispiel werden bei objektrelationalen Datenbanken Abbildungsvorschriften zur Beschreibung, wie XML-Dokumente in Datenbanken gespeichert werden sollen, vorgestellt (die zudem in jedem System anders aussehen); währenddessen wird bei XML-Servern als logisches Datenmodell XML verwendet, weshalb die Darstellung dieser Systeme kompakter ausfallen kann.

Das Kapitel kann einen Stand über die gegenwärtig verfügbaren Systeme und ihre Merkmale geben. Zukünftig sind hier bei allen Systemen rasante Veränderungen und Erweiterungen zu erwarten.

Die Verabschiedung von Empfehlungen des W3Cs hat dabei Auswirkungen auf jedes der vorgestellten Systeme. Die nächste zu erwartende Empfehlung ist XQuery, es ist leicht vorstellbar (und auch schon absehbar), dass die Hersteller von XML-Datenbanken diese Sprache unterstützen werden.

Die objektrelationalen Datenbanksysteme verfolgen momentan alle einen eigenen Weg, um Daten aus einer Datenbank als XML-Dokumente auszugeben und um XML-Dokumente zu speichern und anzufragen. Auch hierfür ist eine Standardisierung zu erwarten. Es gibt eine erste Standardisierungsinitiative SQL/XML, in der ein ANSI-Standard entwickelt werden soll und an dem Vertreter aller großen Herstellerfirmen beteiligt sind. In Zukunft ist hier also eine Vereinheitlichung der Vorgehensweisen zu erwarten [Melt01].

In diesem Kapitel wurden die Merkmale existierender Systeme dargestellt. Tabelle 13-2 fasst die Eigenschaften nochmal zusammen. Damit ist ein Vergleich der Systeme natürlich nur auf qualitativer Ebene möglich. Es wird jedoch deutlich, dass objektrelationale Datenbanksysteme Gemeinsamkeiten aufweisen, etwa wird Ordnungserhaltung nur bei Vorhandensein eines XML-Datentyps unterstützt, daneben erfolgt bei diesen Systemen eine strukturierte Speicherung durch Abbildung von XML-Strukturen auf relationale oder objektrelationale Strukturen.

	DB2 UDB	Oracle 9i	SQL Server 2000	eXcelon	Tamino	Infonyte	POET
XML-Speicherung							
Modell	ORDM	ORDM	RDM	OODM	nativ	DOM	OODM
Schema notwendig	-	-	XDR,DTD	-	-	-	✓
Validierung gegen Schema	✓	✓	✓	✓	✓	✓	✓
als Ganzes	✓	✓	-	-	-	-	-
generisch	-	-	-	✓	✓	✓	-
strukturiert	✓	✓	✓	-	-	-	✓
auch XML-Fragmente	✓	✓	✓	-	-	-	-
Ordnungserhaltung	(✓)	(✓)	-	✓	✓	✓	(✓)
XML-Datentyp	✓	✓	-	-	-	-	-
Indexierung							
Struktur	(✓)	(✓)	-	✓	✓	✓	✓
Werte	✓	✓	✓	✓	✓	✓	✓
Volltext	✓	✓	-	✓	✓	✓	-
XML-Import/Export	✓	✓	✓	✓	✓	-	✓
Sprachschnittstellen							
Anfragesprache	SQL, XPath1.0	SQL, XPath1.0	SQL, XPath1.0	XPath1.0, XQuery	XPath1.0, XQuery	XPath1.0, XQuery	OQL
IR- Funktionalität	✓	✓	-	✓	✓	-	-
Änderungsoperationen	SQL	SQL	Updategram	XUpdate	DOM	DOM	OQL
XSLT	-	✓	✓	✓	✓	✓	-
Werkzeuge, Besonderheiten	IR, DAD-Checker	IR, Schema-validator	XML-Sichten	XML-Editor, Mapping Tool	XML-Editor, Mapping Tool	Einbettung möglich	-

Tab. 13-2: *Eigenschaften der vorgestellten Systeme*

Als Anfragesprache wird von fast allen Systemen aktuell XPath 1.0 verwandt. Viele Hersteller besitzen mittlerweile auch XQuery-Prototypen und werden in naher Zukunft den sich abzeichnenden Standard XQuery 1.0 umgesetzt haben. Einfache Volltextfunktionalität, die zeichenkettenbasiert vorgeht, existiert in den meisten Systemen; Anfragen unter Nutzung von Information-Retrieval-Techniken bieten hingegen nur zwei Systeme über ergänzende Produkte an. Änderungsoperationen sind sehr unterschiedlich in den einzelnen Systemen realisiert, ein Standard ist momentan nicht absehbar.

Alle Systeme bieten zahlreiche Zusatzkomponenten an, von XSLT-Prozessoren bis zu XML-Editoren und Werkzeugen zur Hilfe bei Abbildungsvorgängen und dem Import oder Export von großen Dokumentmengen.

Soll die Leistungsfähigkeit von Systemen bei der Speicherung großer Datenmengen verglichen werden und die Effizienz der verschiedenen Systeme bei der Beantwortung von Anfragen, so ist das nur durch objektive Benchmark-Tests möglich. Wie das erfolgen kann, wird in Kapitel 14 dargestellt.

Literatur

[Cham98] D. Chamberlin. *A Complete Guide to DB2 Universal Database*. Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[Sörm] *Sörman – Content Management Suite (cms) – The Essential Solution for XML/SGML Objects*. http://www.sorman.se/products/cms/.

[Exce] *excelon*. http://www.exceloncorp.com/products/xis/.

[Melt01] J. Melton. (iso-ansi working draft) *xml-related specifications (sql/xml)*, 2001. http: //www.sqlx.org/.

[CFI+00] Micheal Carey, Daniela Florescu, Zachary Ives, Ying Lu, Jayavel Shanmugasundaram, Eugene Shekita and Subbu Subramanian. *XPERANTO: Publishing Object-Relational Data as XML*. In *WebDB*, May 2000.

[Orac] *Oracle Technology Network*. http://www.oracle.com/xml.

[Rys01] M. Rys. *State-of-the-Art XML Support in RDBMS: Microsoft SQL Server's XML Features*. Data Engineering Bulletin, 24(2): 3–11, 2001.

[Schö01] H. Schöning. *Tamino – A DBMS designed for XML*. In Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany. IEEE Computer Society, 2001.

[ScWä00] H. Schöning and J. Wäsch. *Tamino – An Internet Database System*. In C. Zaniolo, P. C. Lockemann, M. H. Scholl, and T. Grust, editors, Advances in Database Technology – EDBT 2000, 7th International Conference on Extending Database Technology, Konstanz, Germany, March 27-31, 2000, Proceedings, volume 1777 of Lecture Notes in Computer Science, pages 383–387. Springer, 2000.

[Tami] *Tamino XML Server*. http://www.softwareag.com/tamino.

14 Benchmarking von XML-Datenbanksystemen

Timo Böhme, Erhard Rahm

Kurzfassung

Zum Vergleich der Leistungsfähigkeit von XML-Datenbanksystemen sind Benchmarks erforderlich, welche den Spezifika der XML-Datenverarbeitung Rechnung tragen. Das Kapitel beschreibt die wesentlichen Anforderungen an geeignete Benchmarks. Ferner werden drei konkrete Benchmarks vorgestellt und miteinander verglichen: XMach-1, Xmark und XOO7.

14.1 Einführung

Wie in den vorangegangenen Kapiteln dargelegt, gibt es eine Reihe unterschiedlicher XML-Datenbanksysteme. Der Bedarf dafür resultiert zum einen aus der zunehmenden Verbreitung von XML als Datenaustauschformat, u.a. in zahlreichen E-Business-Anwendungen. Zum anderen speichern immer mehr Anwendungs- und Systemprogramme ihre Daten in XML ab. Dazu gehören Konfigurationsdateien, Beschreibungsdokumente u.a. Die Verwaltung dieser Daten in einem Dateisystem reicht für viele Anwendungen nicht aus, so dass der Bedarf an XML-Datenbanksystemen sich ständig erhöht.

Die verfügbaren XML-Datenbanksysteme weisen erhebliche Unterschiede bezüglich der zugrunde liegenden Architektur, Funktionalität und interner Realisierung auf. Bei der Auswahl eines Systems für eine bestimmte Anwendung sind diese Aspekte und die daraus resultierende Leistungsfähigkeit (Performance) wesentliche Kriterien. Zur Leistungsbewertung von Computersystemen und Datenbanksystemen existieren zahlreiche Benchmarks. Ziel von XML-Datenbankbenchmarks ist es, in analoger Weise die Leistungsfähigkeit (Performance) unterschiedlicher XML-Datenbanksysteme umfassend und realitätsnah zu bewerten und einen Leistungsvergleich zwischen einzelnen Systemen zu ermöglichen.

Eine Reihe von Arbeiten zur Speicherung von XML in Datenbanken enthalten Geschwindigkeitsmessungen auf Basis selbst definierter Benchmarks [FlKo99] [FlKM00]. Diesen Messungen ist gemein, dass sie nicht detailliert beschrieben sind, wenige spezifische, auf die jeweilige Arbeit zugeschnittene Operationen enthalten und sich damit nicht zum allgemeinen Vergleich eignen. Dem wachsenden Bedarf nach wohldefinierten Benchmarks für XML-DBS tragen drei im Jahr 2001 spezifizierte Benchmarks Rechnung, die in diesem Kapitel beschrieben und gegen-

über gestellt werden sollen. Informationen zu einem erst kürzlich veröffentlichten Benchmark mit Namen »Michigan Benchmark« finden sich in [RPJA02].

Im Folgenden untersuchen wir nach einer allgemeinen Betrachtung der Eigenschaften von Benchmarks die Anforderungen an XML-spezifische Benchmarks. Im Anschluss daran werden die drei XML-Benchmarks in der Reihenfolge ihrer Veröffentlichung beschrieben: XMach-1, Xmark und XOO7. Die Beschreibung erfolgt in einheitlicher Weise und berücksichtigt die jeweilige Anwendungsdomäne, Datenmerkmale, Operationen sowie die Metriken. Danach erfolgt ein Vergleich zwischen den Benchmarks, und abschließend werden Beobachtungen zu bisher vorliegenden Ergebnissen mit dem Benchmark XMach-1 dargelegt.

14.2 Allgemeine Richtlinien für Benchmarks

Die Definition eines Benchmarks resultiert aus dem Bedarf nach einem Werkzeug zum objektiven Leistungsvergleich von Systemen bezüglich eines bestimmten Anwendungsbereichs. Ein aussagekräftiger Benchmark zur generellen Bestimmung der Leistungsfähigkeit von Computersystemen für alle Anwendungsgebiete ist nicht möglich, da je nach Anwendung stark unterschiedliche Anforderungen bestehen und einzelne Hardware- und Software-Komponenten unterschiedlich stark beansprucht werden. Aus diesem Grund existiert eine Vielzahl von Benchmarks, die spezifische Eigenschaften der Computersysteme oder einzelner Komponenten testen, wie z.B. SPEC CPU2000[1] zur Ermittlung der Integer- und Fließkommaleistung von Prozessoren oder BAPCo[2] zur Bestimmung der Leistungsfähigkeit bei der Ausführung von Bürosoftware. Auch im Datenbank-Umfeld können nicht alle Anwendungsbereiche durch einen Benchmark abgedeckt werden. Aus diesem Grund hat u.a. das Herstellergremium TPC (Transaction Processing Performance Council) verschiedene spezialisierte Benchmarks entwickelt, z.B. TPC-C zur Leistungsbewertung klassischer OLTP-Anwendungen (Online Transaction Processing), TPC-H für komplexe Decision-Support-Anfragen und TPC-W für den Einsatz relationaler Datenbanken im Web-Umfeld (E-Commerce)[3]. Es ist also festzuhalten, dass ein »guter« Benchmark auf einen spezifischen Aufgabenbereich (Domäne) fokussiert ist.

Wie in [Gray93] ausgeführt, sollte ein Benchmark neben der Fokussierung auf eine Domäne noch folgende Eigenschaften besitzen:

- *Relevanz*: Der Benchmark sollte die für den Anwendungsbereich wesentlichen Merkmale und Operationen abdecken, um aussagekräftige Ergebnisse zu erzielen.
- *Portabilität:* Der Benchmark sollte einfach auf allen relevanten Rechner-Plattformen implementiert werden können.

1. http://www.specbench.org
2. http://www.bapco.com
3. http://www.tpc.org

- *Skalierbarkeit:* Der Benchmark sollte für unterschiedlich große Konfigurationen eine Leistungsbewertung ermöglichen, von einem einzelnen PC bis hin zu Großrechner-Konfigurationen, für zentralisierte und verteilte Architekturen sowie für unterschiedlich große Datenmengen. Eine solche Skalierbarkeit ist auch wesentlich, um den Benchmark über einen längeren Zeitraum bei starken Verbesserungen in der CPU-Geschwindigkeit, Speichergrößen etc. als Maßstab einsetzen zu können. Damit die Ergebnisse vergleichbar bleiben, sind geeignete Skalierungsfaktoren zu berücksichtigen.
- *Einfachheit:* Damit ein Benchmark akzeptiert wird, muss seine Architektur und Funktionsweise so einfach sein, dass er leicht verständlich, nachvollziehbar und implementierbar ist. Durch diese Eigenschaft sind die Ergebnisse überprüfbar, wodurch sich ihre Glaubwürdigkeit erhöht.

Neben der Leistungsfähigkeit spielt auch die Bewertung der Kosteneffektivität bei der Auswahl eines Systems eine große Rolle. In TPC-Benchmarks wird die Kosteneffektivität durch einen Quotienten aus den Gesamtkosten einer Konfiguration und dem erzielten Leistungswert (z.B. Transaktionen pro Sekunde) bestimmt.

14.3 Betrachtungen zu XML-Datenbankbenchmarks

Nachdem im vorangegangenen Abschnitt allgemeine Kriterien für Benchmarks diskutiert wurden, geht es jetzt um die speziellen Anforderungen und Probleme von XML-Datenbankbenchmarks. Zunächst wird die Bedeutung der Domänen im XML-Umfeld betrachtet. Anschließend werden Kriterien für den XML-Datenbankbereich besprochen, die den Benchmark grundlegend bestimmen. Am Ende dieses Abschnitts wird auf Problembereiche, die sich aus dem aktuellen Entwicklungsstand von XML ergeben, eingegangen.

14.3.1 Domänen für XML-Datenverwaltung

Ebenso wie es keinen universellen Datenbankbenchmark gibt, kann es auch für die XML-Datenverwaltung keinen alle Anwendungsbereiche abdeckenden, universellen Benchmark geben. Aufgrund des breiten Einsatzspektrums von XML und der hohen Bandbreite unterschiedlicher XML-Daten und den damit verbundenen Operationen gilt dies hier sogar in verstärktem Maße. XML-Datenbankbenchmarks sind daher auf einen bestimmten Anwendungsschwerpunkt auszurichten. Für diese Domänen-Fokussierung wesentlich sind vor allem die Art der XML-Daten (dokumentorientiert, datenorientiert oder Mischformen) sowie die jeweils benötigten Operationen. Im Folgenden diskutieren wir die für die Benchmarks relevanten Merkmale / Anforderungen, die sich aus den drei wesentlichen Arten von XML-Daten ergeben.

Dokumentenkollektionen

Ein wesentliches Anwendungsgebiet für XML-Datenbanken ist die Verwaltung von *dokumentorientierten* XML-Daten (vgl. Kapitel 5) im Rahmen von Dokumentenkollektionen, wie sie in Firmen-Intranets oder im Internet auftreten. Die Elementhierarchie in den Dokumenten ist typischerweise bis zu 10 Stufen geschachtelt, und die Dokumentgrößen reichen von einigen Kilobyte bis zu wenigen Megabyte.

Für diese Domäne sind folgende Operationstypen relevant:

- Hierarchische und sequenzielle Navigation
- Volltextsuche auf Elementinhalten
- Abrufen von vollständigen Dokumenten
- Extraktion von Dokumentfragmenten
- Erstellung von Verzeichnissen ausgewählter Elemente (z.B. Inhaltsverzeichnis)
- Einfügen und Löschen von Dokumenten
- Einfügen und Löschen von Dokumentfragmenten in einem Dokument

Aufgrund der unregelmäßigen und heterogenen Strukturen der Dokumente treten bei den Anfrageoperationen typischerweise auch Wildcards in den Pfadausdrücken auf.

Strukturierte Daten

Im Gegensatz zur vorangegangenen Domäne sind *datenorientierte* oder strukturierte XML-Daten zu verwalten, wie sie u.a. in zahlreichen E-Business-Anwendungen dominieren. Die Elementhierarchie ist typischerweise flach und beträgt selten mehr als fünf Stufen. Da die XML-Datenobjekte (Dokumente) vergleichbar mit den Sätzen relationaler Datenbanken sind, liegt ihre Größe meist im Bereich von einigen hundert Byte bis zu wenigen Kilobyte.

Bei den Operationen ergeben sich im Gegensatz zur vorher beschriebenen Domäne folgende Schwerpunkte:

- Suche auf Attributwerten
- Anfragen mit Sortierung, Aggregation und Verbundanweisungen
- Erzeugung neuer Ergebnisdokumente basierend auf Anfragedaten
- Einfügen und Löschen von Dokumenten
- Änderung von Attributwerten

Die XML-Daten sind in der Regel gemäß einem oder wenigen vordefinierten Schemata strukturiert. Damit besteht bei der Definition der Anfragepfade kaum Bedarf für Wildcards, wodurch die Anfragen ein höheres Maß an Selektivität aufweisen. In Abhängigkeit von der verwendeten Schemasprache und den von der Datenbank beim Speichern unterstützten Datentypen sind Datentyptransformationen (Casting) bei der Anfrageverarbeitung notwendig. Diese aufwändigen Umwandlungen sollten in eigenen Operationstypen untergebracht werden, damit die Ergebnisse der anderen Operationstypen keine Verzerrung erfahren.

Gemischt strukturierte Daten

Die beiden beschriebenen Ansätze stellen Extreme beim Einsatz von XML dar – dokumentorientiert vs. datenorientiert. Viele Anwendungsgebiete erfordern jedoch Mischformen von XML-Daten. Zum Beispiel sind in Online-Katalogen neben strukturierten Produktdaten auch textuelle bzw. multimediale Zusatzinformationen zu verwalten.

Für solche Anwendungsbereiche kann die Kombination der verschiedenen Eigenschaften von XML-Daten unterschiedlich realisiert werden. Eine Möglichkeit besteht darin, dass die Elementhierarchie der XML-Daten durch ein Schema wohldefiniert ist, wobei es jedoch einzelne Elemente gibt, deren Inhalt nicht beschränkt ist und die damit die dokumentorientierten Daten aufnehmen können. In einer anderen Variante werden datenorientierte und dokumentorientierte XML-Daten in separaten Dokumenten/Objekten gespeichert und durch Verweise wird die Verbindung zwischen Dokumenten der beiden Klassen hergestellt.

Die für diese Domäne relevanten Operationstypen sind ebenfalls eine Mischung der Operationen für datenorientierte und dokumentorientierte XML-Daten. Ein Benchmark muss dabei das Verhältnis der Operationstypen entsprechend dem simulierten Anwendungsfall wählen. Eine Übernahme aller Operationstypen würde zu einem sehr umfangreichen Benchmark führen, der das Kriterium der Einfachheit verletzt. Zudem sind nicht alle Operationen für eine Domäne gleichermaßen bedeutsam.

14.3.2 Spezifische Kriterien für einen XML-Datenbankbenchmark

Neben der Festlegung der Domäne sind bei einem XML-Datenbankbenchmark weitere Zielsetzungen festzulegen, in denen sich die bisherigen Vorschläge auch unterscheiden:

- *Skopus: Bewertung des gesamten DBS oder einzelner Komponenten*
  Aus Sicht der Nutzer eines XML-DBS ist das Leistungsverhalten des Gesamtsystems ausschlaggebend, das sich aus dem Zusammenwirken aller Komponenten ergibt. Die Bewertung wichtiger Einzelkomponenten, insbesondere des Anfrageprozessors, kann jedoch auch schon aussagekräftige Leistungskennzahlen ergeben. Da bei der Bewertung des Anfrageprozessors mehr die Effizienz bezüglich bestimmter Operationen im Vordergrund steht, ist die gewählte Anwendungsdomäne von geringerer Bedeutung.
- *Ein- vs. Mehrbenutzerbetrieb*
  Datenbanksysteme sind generell für die gleichzeitige Nutzung durch mehrere Benutzer ausgelegt, so dass auch Benchmarks für XML-Datenbanksysteme Leistungsaussagen für den Mehrbenutzerbetrieb liefern sollten. Wenn es jedoch um die Evaluierung einzelner Komponenten geht, z.B. des Anfrageprozessors, so kann die Beschränkung auf den Einzelbenutzerbetrieb sinnvoll

sein. In diesem Fall sind nur Antwortzeitbewertungen möglich, jedoch keine Messung der Durchsatzleistung.

- *Operationen-Mix*
  Die Bestimmung der zu messenden Last erfordert die Festlegung einer bestimmten Anzahl von Operationen unterschiedlicher Art und Komplexität. Die Anzahl der Operationen sollte eher gering sein, um einen überschaubaren Vergleich der Systeme zu ermöglichen. Versucht man dennoch, möglichst alle relevanten Operationstypen abzudecken, führt dies zu relativ komplexen Operationen. Je mehr Funktionalität in einer Operation verwendet wird, um so geringer ist die Aussagekraft der Laufzeitmessung, da z.B. Rückschlüsse auf die einzelnen Anfrageprozessorkomponenten nicht mehr möglich sind. Ein optimales Verhältnis dieser beiden Einflussgrößen, Anzahl und Komplexität, kann nur in Abhängigkeit von den intendierten Zielen des Benchmarks bestimmt werden.

14.3.3 Problembereiche von XML-Datenbankbenchmarks

Das frühe Entwicklungsstadium der XML-Datenbanksysteme, die unterschiedlichen Architekturen und unterstützten Funktionalitäten können zu Problemen führen, die bei der Spezifikation der Benchmarks beachtet werden müssen.

Das Hauptproblem ist das Fehlen einer standardisierten Anfragesprache. Obwohl mit XQuery eine solche in der Spezifikationsphase ist, wird es noch längere Zeit dauern, bis diese durch den Großteil der XML-Datenbanksysteme unterstützt wird. Als kleinster gemeinsamer Nenner steht in den meisten Fällen XPath 1.0 für Anfrageoperationen zur Verfügung, womit jedoch nur ein Teil der benötigten Operationen umgesetzt werden kann. Auch die proprietären Anfragesprachen der verfügbaren XML-Datenbanksysteme sind nur selten mächtiger als XPath. Komplexere Operationen müssen deshalb durch API-Zugriffe auf die Daten, z.B. mittels DOM, realisiert werden. Bei der Definition eines Benchmarks ist man deshalb gezwungen, die Operationen als beschreibenden Text oder beispielhaft in der aktuellen Entwurfsfassung von XQuery anzugeben.

Gibt es für die Anfragesprache mit XPath wenigstens eine allgemein akzeptierte Grundlage, so existiert für die Datenmanipulation nichts Vergleichbares. Selbst in XQuery sind bisher keine ändernden Operationen vorgesehen. Mit XUpdate[4] liegt zwar ein Arbeitspapier der XML:DB-Initiative vor, jedoch wird dieser Vorschlag bisher noch von keinem am Markt befindlichen XML-DBS umgesetzt. Ein weiteres Problem mit Änderungsoperationen ist, dass einige XML-Datenbanken nicht in der Lage sind, einzelne Fragmente von XML-Daten zu manipulieren. Um Änderungen durchzuführen, müssen dort die Dokumente vollständig ausgelesen, entfernt, geändert und wieder eingefügt werden. Zur Wahrung der Portabilitätseigenschaft ist es deshalb notwendig zu entscheiden, ob der

4. http://www.xmldb.org/xupdate/index.html

Benchmark Änderungsoperationen enthalten soll und welchen maximalen Umfang die zu ändernden Dokumente besitzen dürfen.

Die Portabilität muss auch bei der Zusammenstellung der Datenbankanfragen beachtet werden, da sich der angebotene Funktionsumfang der einzelnen Datenbanken derzeit noch beträchtlich unterscheidet. Portabilität und Relevanz arbeiten in diesem Fall also gegenläufig. Die Spezifikation zahlreicher Operationen erfolgt auf Kosten der Portabilität; umgekehrt kann ein hochgradig portabler Benchmark nur einen kleinen Teil der relevanten Operationen enthalten. Um kein System auszuschließen, kann der Benchmark alle Operationen einzeln und unabhängig voneinander testen und besteht demzufolge aus einer Anzahl von Sub-Benchmarks. Dies führt zu zahlreichen Einzelergebnissen, welche die Vergleichbarkeit zwischen Systemen erschwert. Alternativ kann der Benchmark vorsehen, fehlende Funktionen im zu testenden System zu ergänzen (ähnlich wie beim Einsatz des Systems in der Praxis, wobei fehlende Funktionen anwendungsseitig realisiert werden müssten).

14.4 XMach-1[5]

An der Universität Leipzig wurde im Jahr 2000 mit der Entwicklung von XMach-1 (*X*ML Data *Ma*nagement ben*ch*mark) begonnen, dessen Spezifikation Anfang 2001 veröffentlicht wurde [BöRa01]. Es war der erste XML-Datenbankbenchmark mit Anspruch auf allgemeine Anwendbarkeit. Die wesentlichen Zielsetzungen von XMach-1 sind Skalierbarkeit, Mehrbenutzerbewertung und die Evaluierung des gesamten Datenverwaltungssystems. Im Gegensatz zu den anderen XML-Benchmarks enthält er eine genaue Beschreibung der Testumgebung sowie Vorgaben zur Ausführung der Operationen.

Domäne

Der Benchmark simuliert eine Web-Anwendung zur Verwaltung unterschiedlicher textorientierter Dokumente. Das Testsystem kann beliebig viele Datenbank- sowie Applikations-Server umfassen, wobei letztere die generischen Anfragen der Clients in Anfragen auf die Datenbank abbilden und dabei anfallende Mapping- und Transformationsaufgaben übernehmen können. In großen Teilen entspricht dieses Szenario der in Kapitel 14.3.1 beschriebenen Domäne »Dokumentenkollektion«. Durch das Verwalten von Metadaten der Kollektion in einem strukturierten Dokument enthält der Benchmark jedoch auch Aspekte des Typs »Strukturierte Daten« und ist demzufolge der Domäne »Gemischt strukturierte Daten« mit Schwerpunkt auf dem dokumentorientierten Aspekt zuzuordnen.

5. Homepage: http://dbs.uni-leipzig.de/de/projekte/XML/XmlBenchmarking.html

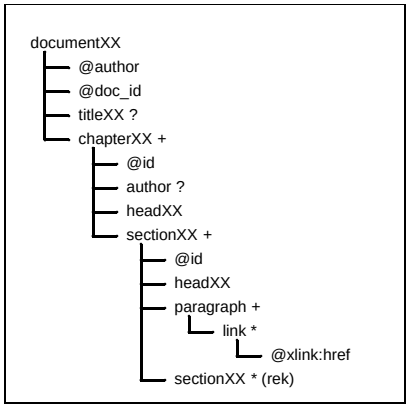


Abb. 14-1: *DOM-Struktur eines Textdokumentes in XMach-1*

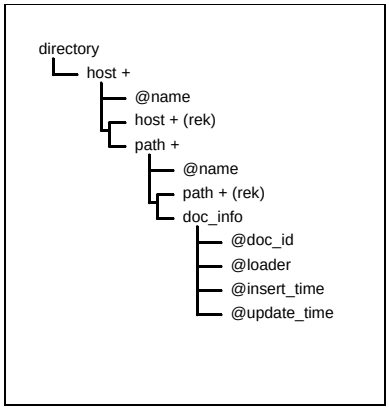


Abb. 14-2: *DOM-Struktur des Metadatendokumentes*

Daten

Der Benchmark enthält zwei Dokumenttypen. Der größte Teil der Daten besteht aus Dokumenten, die in ihrer Struktur und Inhalt Textdokumenten wie Büchern, Aufsätzen o.ä. nachempfunden sind und damit vom Typ dokumentorientiert sind. In Abb. 14-1 ist die DOM-Hierarchie der Elemente und Attribute dargestellt. Diese Dokumente werden durch einen parametrisierbaren Generator synthetisch erzeugt. Um realitätsnahe Ergebnisse beim Speichern und Anfragen der Textinhalte zu erreichen, bestehen diese aus den 10.000 häufigsten englischen Wörtern, wobei ihre Verteilung der in natürlich sprachigem Text entspricht. Die Größe der Dokumente variiert zwischen 2 und über 100 Kilobyte mit einem Mittelwert von ca. 16 Kilobyte, so dass typische Dokumentgrößen ohne grafische oder multimediale Inhalte von dem Benchmark abgedeckt werden. Die Variabilität der Dokumentgröße wird durch die freie Anzahl von *chapter*- und *section*-Elementen sowie die rekursive Definition von *section* erreicht. Durch geeignete Parametrisierung bei der Generierung der Dokumente werden sowohl flache Dokumente (viele *chapter*-Elemente, wenige geschachtelte *section*-Elementen) als auch tiefe Dokumente (hohe Anzahl geschachtelter *section*-Elemente) erzeugt.

Den gemischt strukturierten Charakter erhält der Benchmark durch ein Dokument, das Metadaten über alle enthaltenen Dokumente enthält, wie z.B. URL, Name, Einfüge- und Änderungszeit. Abb. 14-2 zeigt die DOM-Struktur für dieses Dokument. In ihm sind alle Informationen in den Attributen enthalten und die Reihenfolge von Geschwisterelementen ist beliebig, so dass es vom Typ datenorientiert ist. Im Unterschied zu festen relationalen Strukturen besitzt es auch semistrukturierte Eigenschaften, die sich aus der variablen Pfadtiefe und der Optionalität von Attributen ergeben. Die Größe des Dokumentes ist abhängig von der Anzahl der Textdokumente. Jeder Eintrag benötigt ca. 160 Byte.

Der Benchmark unterstützt sowohl eine schemalose Speicherung der Dokumente als auch eine Variante mit Schemanutzung. Ein wesentliches Merkmal ist dabei die unbegrenzte Zahl von Tag-Namen, um zu testen, wie gut XML-Datenbanksysteme in der Lage sind, viele verschiedene Strukturen bzw. Schemata zu verwalten. Diesem Aspekt wird im Benchmark durch die Verwendung eines Zählers an bestimmten Tag-Namen (siehe Abb. 14-1) Rechnung getragen. Damit werden pro Schema durchschnittlich 20 Dokumente generiert.

Die Skalierbarkeit des Benchmarks bzgl. des Datenvolumens wird durch die Erhöhung der Anzahl der Dokumente in Vielfachen von 10 erreicht. Dadurch wächst auch das Metadatendokument proportional. Das Verhältnis von Dokumenten pro Schema, Anzahl von unterschiedlichen Autoren pro Dokumentanzahl u.a. wird durch die Dokumentanzahl nicht beeinflusst, so dass die Benchmarkresultate vergleichbar bleiben.

Operationen

XMach-1 definiert acht Anfrage- und drei Änderungsoperationen, die in Tabelle 14-1 aufgelistet sind. Die Operationen reichen vom Abfragen komplexer vollständiger Dokumente (Q1) über Textsuche (Q2) bis hin zu Abfragen mit Sortier- und Verbundoperatoren, d.h., es liegt auch hier eine Mischung von dokument- und datenorientierten Anfragen vor. Die Änderungsoperationen beinhalten das Einfügen und Löschen von Textdokumenten (M1, M2) sowie die Änderung von Attributwerten des Metadatendokuments (M3). Für die Operationen liegt neben der verbalen Beschreibung auch eine Spezifikation in XQuery vor. Zur Illustration der unterschiedlichen Anfragekomplexität sind in Tabelle 14-1 für die Operationen Q3 und Q4 die XQuery-Ausdrücke angegeben. Während in Q4 die Anfrage im Wesentlichen aus einem regulären Pfadausdruck besteht, muss zur Formulierung von Q3 eine rekursive benutzerdefinierte Funktion eingesetzt werden.

Die Operationen werden gemeinsam in einem Mix ausgeführt, wobei die prozentualen Anteile der einzelnen Operationen festgelegt sind. Entsprechend der simulierten Domäne besitzt die Anfrage Q1 mit 30% das größte Gewicht. Dagegen kommt den Änderungsoperationen mit teilweise weniger als 1% nur geringe Bedeutung zu. Sie haben jedoch einen nicht unerheblichen Einfluss auf die Ausführung der Anfrageoperationen, da das Cache- und Transaktionsmanagement für die Aktualität der Daten sorgen muss.

Aufgrund der unterschiedlichen Schemata der Dokumente müssen einige Anfragen mit Wildcards arbeiten, was besondere Anforderungen an den Anfrageoptimierer stellt. Wegen der Operation M3 kann der Benchmark nur vollständig implementiert werden, wenn das Datenbanksystem die Änderung von Dokumentfragmenten erlaubt.

ID	Operation	Kommentar
Q1	Liefere das Dokument zu URL X.	Ausgabe eines vollständigen Dokumentes. Selektion des Dokumentes mittels Verbund mit Metadatendokument.
Q2	Liefere doc_id der Dokumente, die die Phrase X in einem paragraph-Element enthalten.	Testet Leistungsfähigkeit der Volltextsuche.
Q3	Navigiere beginnend beim ersten chapter-Element über jedes erste section-Element. Gib das letzte section-Element zurück.	Simuliert Navigation über den Dokumentbaum mittels Sequenzoperatoren.

	DEFINE FUNCTION deepestFirstSection(ELEMENT $e) RETURNS ELEMENT
	{
	IF (empty($e/section10)) THEN $e
	ELSE deepestFirstSection($e/section10[1])
	}
	deepestFirstSection(/document10[@doc_id="d1"]/chapter10[1]/section10[1])	

Q4	Liefere eine flache Liste der head-Elemente aller section-Elemente des durch doc_id X selektierten Dokumentes.	Rekonstruktion von Teilen des Dokumentes. Die Erstellung eines Inhaltsverzeichnisses wird simuliert.
	FOR $a IN /document10[@doc_id="d1"]//section10/head10 RETURN $a	
Q5	Liefere alle Dokumentnamen (name-Attribut des letzten path-Elementes), die unterhalb einer gegebenen URL liegen.	Hier kann der Einfluss der Elementordnung geprüft werden.
Q6	Finde alle author-Elemente mit Inhalt X und liefere id-Attribut des Elternelementes.	Selektion über Elementinhalt.
Q7	Liefere doc_id von allen Dokumenten, die wenigstens X-mal referenziert werden.	Gruppierung und Zählerfunktionalität werden hier getestet.
Q8	Liefere doc_id von den 100 zuletzt geänderten Dokumenten, die ein author-Attribut besitzen.	Benötigt eine Reihe von datenorientierten Funktionen: Zähler, Sortierung, Verbund, Existenzoperator.
M1	Füge neues Dokument ein.	Testet die Einfügeleistung komplexer Dokumente mit aktiven Indexen.
M2	Lösche Dokument mit doc_id X.	Testet das Löschen eines komplexen Dokumentes mit aktiven Indexen.
M3	Ändere Namen und update_time-Attribut des Dokumentes mit doc_id X.	Testet die Effizienz von Änderungsoperationen auf Attributwerten.

Tab. 14-1: *Operationen von XMach-1 mit ausgewählten XQuery-Beispielen*

Metriken

Als Mehrbenutzer-Benchmark steht die Messung des Durchsatzes im Vordergrund. In XMach-1 wird diese in *Xqps (XML queries per second)* gemessen und entspricht der Anzahl der durchgeführten Q1-Anfragen. XMach-1 sieht eine Un-

terscheidung in eine schemaunterstützte und eine schemalose Variante vor und bietet dafür zwei verschiedene Einheiten an. Neben dem Durchsatz werden auch die Antwortzeiten der einzelnen Operationen gemessen. Durch Vorgeben der maximalen Antwortzeiten für jeweils 90% der Anfragen wird erreicht, dass die Anzahl der Clients für einen maximalen Durchsatz nicht zu Lasten der Antwortzeiten optimiert werden kann. Auf der anderen Seite wird durch Denkzeiten auf der Seite der Clients erreicht, dass zur Erhöhung des Durchsatzes die Anzahl der Clients steigen muss, so dass ein Mehrbenutzerbetrieb erzwungen wird. Die Skalierung auf leistungsfähigere Systeme kann demzufolge sowohl durch die Anzahl der Dokumente als auch durch die Anzahl der Clients erfolgen.

Die Erzielung eines einzelnen Ergebniswertes pro System gestattet die Bestimmung der Kosteneffektivität ($ bzw. Euro pro Xqps) analog zu TPC-Benchmarks. Selbstverständlich können mit dem Benchmark jedoch auch Einbenutzerbewertungen vorgenommen werden, insbesondere zu den Antwortzeiten der einzelnen Operationen. Außerdem werden Angaben bestimmt zur Ladezeit der Datenbank und zur Größe der Datenbank und von Indexstrukturen.

14.5 Xmark[6]

Dieser Benchmark wurde am National Research Institute for Mathematics and Computer Science (CWI) der Niederlande entwickelt und Mitte 2001 in der Version 1.0 veröffentlicht [SWKF01]. Hauptziel des Benchmarks ist die Untersuchung der Performance des Anfrageprozessors, welches sich in der großen Anzahl der Anfragen und in der Beschränkung auf ein lokales Computersystem (1 Rechner) widerspiegelt. Die dem Benchmark zugrunde liegenden Richtlinien werden in [SWKF01b] ausführlich diskutiert.

Domäne

Wie schon bei XMach-1 liegt auch diesem Benchmark ein konkretes Anwendungsszenario zugrunde. Modelliert wird die Datenbank eines Internet-Auktionsportals. Diese, als ein großes XML-Dokument gestaltete Datenbank, enthält eine Anzahl von Fakten, die durch ihre feste Struktur im Wesentlichen datenorientierte Eigenschaften aufweisen. Durch die Aufnahme von Beschreibungstexten werden jedoch auch dokumentorientierte Aspekte eingebracht, d.h. auch dieser Benchmark gehört zur Domäne »Gemischt strukturierte Daten«, hier jedoch mit Schwerpunkt auf dem datenorientierten Aspekt.

Daten

Der Benchmark verwaltet alle Daten innerhalb eines einzigen Dokumentes, dessen DOM-Struktur in Abb. 14-3 wiedergegeben ist. Es enthält die Auktionsdaten als

6. Homepage: http://monetdb.cwi.nl/xml/index.html

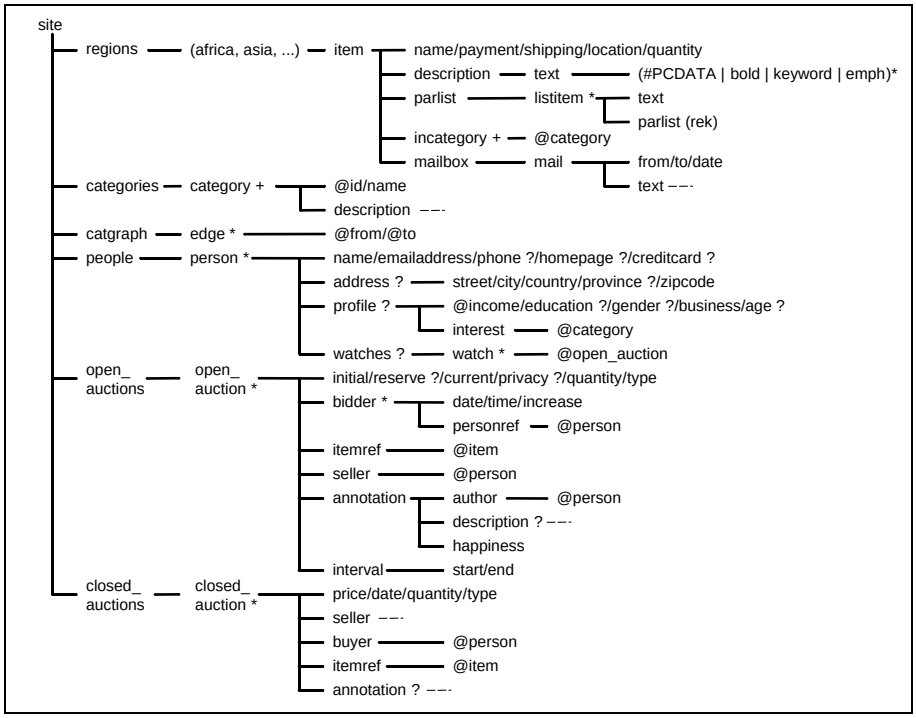


Abb. 14-3: *DOM-Struktur des Xmark-Dokumentes*

strukturierte Fakten, welche in den Hauptinhalten (Kindelemente des Wurzelelementes) Gegenstände (*item*) – unterteilt nach Regionen –, Kategorien (*categories*), Personen (*people*), offene und geschlossene Auktionen (open_auctions / closed_ auctions) gespeichert sind. Die sich daraus ergebenden Teilbäume sind durch zahlreiche Verweise miteinander verbunden. Die Elementreihenfolge von Geschwisterelementen ist größtenteils frei. Nur bei einzelnen Elementen, wie z.B. der Bieterhistorie (*bidder*) oder der Beschreibung der Gegenstände (*description*) ist die Reihenfolge relevant.

Die Größe des Dokumentes kann über einen Skalierungsfaktor von 10 Megabyte bis 10 Gigabyte eingestellt werden. Skaliert wird über die Anzahl von Objekten ausgewählter Mengen, z.B. Gegenstände und Personen, indem deren Basisanzahl mit dem Skalierungsfaktor multipliziert wird. Dieses hat jedoch zur Folge, dass sich das Verhältnis bestimmter Elemente zueinander ändert.

Das Dokument weist eine recht komplexe, jedoch auch feste Struktur auf. Die Tatsache, dass sich bei der Skalierung zwar die Anzahl der Elemente, jedoch nicht die Anzahl der Elementtypen erhöht, unterstreicht den stark ausgeprägten datenorientierten Skopus des Benchmarks.

Operationen

Entsprechend seiner Ausrichtung auf den Anfrageprozessor definiert Xmark eine große Anzahl von Anfragen (20), die jeweils verschiedene Aspekte der Anfrageverarbeitung testen. Die Anfragen sind in Tabelle 14-2 aufgelistet. Neben der Formulierung der Anfragen in textueller Form liegen diese auch in XQuery-Syntax vor. Für ein besseres Verständnis der Operationsbeschreibungen sind in der Tabelle für die Operationen Q3 und Q20 auch die XQuery-Umsetzungen angegeben. Da ein entsprechender Standard zur Datenmanipulation noch nicht vorliegt, verzichtet Xmark auf die Evaluierung von Änderungsoperationen.

Die in der Tabelle angegebenen Bereiche entsprechen der Einordnung der Operationen durch die Autoren des Benchmarks. Man erkennt daran den Skopus auf die Bewertung bestimmter Aspekte der Anfrageverarbeitung. Die relativ große Zahl der Operationen resultiert auch aus der Aufnahme mehrerer funktional ähnlicher Anfragen (z.B. Verbundanfragen Q8/Q9, Q11/Q12), um die Unterstützung bestimmter Optimierungsmöglichkeiten zu bewerten. Einige Anfragen wie z.B. Q10 sind nur für kleinere Datenbestände sinnvoll. Eine alternative Einteilung der Operationen nach »dokumentorientiert« vs. »datenorientiert« ist prinzipiell möglich, jedoch gibt es Überlappungen. Anfragen, die primär zum dokumentorientierten Bereich gehören, sind z.B. Q2-Q4 (Elementreihenfolge), Q13 (Rekonstruktion, Elemente mit gemischtem Inhalt) und Q14 (Textsuche). Dem datenorientierten Bereich sind besonders die Anfragen Q1, Q11, Q12 und Q20 zuzuordnen, da diese hauptsächlich auf Attributwerten arbeiten. Die anderen Anfragen enthalten Aspekte, die für beide Bereiche relevant sind.

Metrik

Die Ausführung der Anfragen erfolgt im Einbenutzerbetrieb, wobei jede Operation einzeln betrachtet wird. Als Resultat liefert der Benchmark die Ausführungszeiten für die einzelnen Operationen. Bei Operationen mit sehr kurzen Ausführungszeiten wird ein Wiederholungsfaktor angegeben, der bestimmt, wie oft die Operation nacheinander ausgeführt wird. In diesem Fall enthält die Ausführungszeit alle Wiederholungen. Ein Problem hierbei ist, dass durch ein von einigen Systemen vorgenommenes Caching von Anfrage-Antwort-Paaren die durchschnittliche Antwortzeit vom Ausmaß der Caching-Effekte beeinflusst wird.

Bereich	ID	Operation	Kommentar
Genaue Übereinstimmung	Q1	Liefere den Namen des Gegenstandes, registriert in Nordamerika, mit der ID 'item20748'.	Einfache Anfrage mit kompletter Pfadangabe und Zeichenkettensuche.
Elementreihenfolge	Q2	Liefere das erste Gebot für alle offenen Auktionen.	Testet Kosten für Zugriff unter Beachtung der Elementreihenfolge.
	Q3	Liefere das erste und das aktuelle Gebot aller offenen Auktionen, bei denen das aktuelle Gebot wenigstens doppelt so hoch wie das erste ist.	Komplexere Anwendung des Zugriffs auf bestimmte Elementpositionen.
		FOR $b IN document("auction.xml")/site/open_auctions/open_auction WHERE $b/bidder/[0]/increase/text()*2 <= $b/bidder[last()]/increase/text() RETURN <increasefirst=$b/bidder[0]/increase/text() last=$b/bidder[last()]/increase/text()	
	Q4	Liste die Mindestgebote aller offenen Auktionen, bei denen eine bestimmte Person A vor einer anderen Person B ein Gebot abgegeben hat.	Testet Aufwand zur Feststellung der Tag-Reihenfolge.
Casting	Q5	Wie viele verkaufte Gegenstände kosten mehr als 40?	Zur Abarbeitung der Anfrage muss der als Zeichenkette gespeicherte Preis in eine Zahl umgewandelt werden.
Reguläre Pfadausdrücke	Q6	Wie viele Gegenstände sind für alle Regionen gelistet?	Testet einfache Pfadausdrücke mit Wildcards.
	Q7	Wie viele Texte (description-, mail- und email-Elemente) gibt es?	Testet den Optimierer für Pfadausdrücke mit Wildcards. Ein kompletter Baumdurchlauf sollte nicht durchgeführt werden.
Referenzverfolgung	Q8	Liste die Namen aller Personen und die Anzahl der von ihnen gekauften Gegenstände.	Stellt mittels Verweisen einen Verbund zwischen Personen und geschlossenen Auktionen her.
	Q9	Liste die Namen aller Personen und die Namen der Gegenstände, die sie in Europa gekauft haben.	Zusätzlich zu Q8 werden noch Gegenstände über Verweise in den Verbund aufgenommen.
Komplexe Ergebnisse	Q10	Liste alle Personen nach ihren Interessen.	Erstellt umfangreich strukturiertes Ergebnisdokument.
Verbund über Attributwerte	Q11	Liste für jede Person die Anzahl der derzeit angebotenen Gegenstände, deren Preis unter 0,02% des Einkommens der Person ist.	Der Verbund über Attributwerte ist umfangreicher als bei dem Verbund mittels Referenzen.
	Q12	Liste für jede Person mit einem Einkommen über 50.000 die Anzahl der zum Verkauf angebotenen Gegenstände auf, deren Preis kleiner als 0,02% des Einkommens ist.	Ähnlich zu Q11 mit zusätzlicher Selektion der Personen. Beide Operationen bieten Optimierern viele Angriffspunkte.

Tab. 14-2: *Operationen von Xmark*

Bereich	ID	Operation	Kommentar
Rekonstruktion	Q13	Liste den Namen und die Beschreibung aller für Australien angebotenen Gegenstände auf.	Hier müssen größere Teile des Dokumentes wiederhergestellt werden.
Volltextsuche	Q14	Liefere die Namen aller Gegenstände, deren Beschreibung das Wort 'gold' enthält.	Kombiniert strukturelle Suche mit Volltextsuche.
Pfadnavigation	Q15	Liefere die hervorgehobenen Schlüsselwörter in Anmerkungen von geschlossenen Auktionen.	Evaluiert lange Pfadausdrücke ohne Wildcards.
	Q16	Liefere die IDs der Verkäufer von jenen geschlossenen Auktionen, die wenigstens ein hervorgehobenes Schlüsselwort in den Anmerkungen haben.	Test auf Existenz mit langen Pfadausdrücken ohne Wildcards.
Fehlende Elemente	Q17	Welche Personen haben keine Homepage?	Sucht nach fehlenden Elementen.
Funktionsaufruf	Q18	Konvertiere die Währung der Mindestgebote aller offenen Auktionen.	Testet die Ausführung benutzerdefinierter Funktionen (UDF).
Sortierung	Q19	Liefere eine alphabetisch sortierte Liste aller Gegenstände zusammen mit der Ortsangabe.	Sortierung einer großen Anzahl von Zeichenketten.
Aggregation	Q20	Gruppiere alle Kunden nach ihrem Einkommen in 4 Gruppen und gib die Kardinalität jeder Gruppe aus.	Pro Gruppe wird eine Selektion über dem income-Attribut vorgenommen. Diese Wiederholung der Operation kann für Optimierungen genutzt werden.

```
<result>
 <preferred>
  COUNT (document("auction.xml")/site/people/person/\
        profile[@income >= 100000])
 </preferred>,
 <standard>
  COUNT (document("auction.xml")/site/people/person/\
        profile[@income < 100000 and @income >= 30000])
 </standard>,
 <challenge>
  COUNT (document("auction.xml")/site/people/person/\
        profile[@income < 30000])
 </challenge>,
 <na>
  COUNT (FOR$p in document("auction.xml")/site/people/person
        WHEREEMPTY($p/@income)
        RETURN$p)
 </na>,
</result>
```

Tab. 14-2: *Operationen von Xmark*

14.6 XOO7[7]

Ebenfalls Mitte 2001 wurde der Benchmark XOO7 vorgestellt, welcher an der National University of Singapore entwickelt wurde. Im Gegensatz zu den vorangegangenen Benchmarks ist er keine komplette Neuentwicklung, sondern basiert auf dem Benchmark OO7 [CaDN93], der zur Evaluierung objektorientierter Datenbanken spezifiziert wurde. Obwohl die Autoren des Benchmarks auf Parallelen des Objektmodells und des XML-Datenmodells hinweisen, zeigt es sich, dass die Datenstruktur und auch die Operationen von OO7 nur einen Teil der Aufgaben eines XML-Datenbanksystems abdecken. Diesen Mangel versuchen die Autoren durch eine Ergänzung der Datenstruktur und zusätzliche Anfragen zu kompensieren.

Domäne

Im Unterschied zu den beiden anderen Benchmarks liegt XOO7 kein konkretes Anwendungsszenario zugrunde. Es werden vielmehr generische Beschreibungen komplex aufgebauter Module mit im Wesentlichen »Teil-von«-Beziehungen modelliert. Dies geschieht in einer festen und regelmäßigen Struktur, bei der alle Daten in den Attributen liegen, womit das Dokument datenorientierte Eigenschaften besitzt. Durch die Aufnahme von textuellen Beschreibungen und entsprechenden Operationen darauf, erhält der Benchmark auch dokumentorientierte Aspekte und ist demnach, wie schon die vorangegangenen Benchmarks, der Domäne »Gemischt strukturierte Daten« zuzuordnen, wobei der datenorientierte Anteil dominiert.

Daten

Wie in Xmark gibt es auch bei XOO7 nur ein einziges Dokument, welches alle Daten umfasst. In Abb. 14-4 sind die Elementhierarchie sowie die zugehörigen Attribute dargestellt. Betrachtet man das Schema des zugrunde liegenden objektorientierten Benchmarks, wird dessen Ausrichtung auf Vererbung und Objektreferenzierung ersichtlich. Während das erste Konzept in XML nicht vorhanden ist, und deshalb für die Elemente die Attribute dupliziert wurden, wird auch die Möglichkeit der Referenzierung via ID und IDREF nicht genutzt, so dass die einzelnen »Objekte« redundant in den Daten vorliegen. Die enthaltenen Querverweise können somit nicht durch das Datenbanksystem verwaltet werden.

Die Daten können mittels eines Generators erzeugt werden, der durch Vorgabe mehrerer Parameter zur Rekursionstiefe und Anzahl von Kindelementen Dokumente in verschiedenen Größen erzeugen kann. Um eine Vergleichbarkeit von Ergebnissen zu ermöglichen, gibt es Vorgaben zu Parameterkonfigurationen, die als Small, Medium und Large bezeichnet sind und zu Dokumentgrößen von wenigen Megabyte bis zu einem Gigabyte reichen. Die Generierung der Zeichen-

7. Homepage: http://www.comp.nus.edu.sg/~ebh/XOO7.html

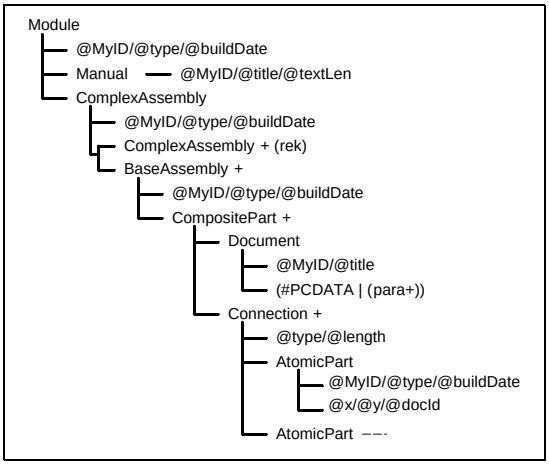


Abb. 14-4: *DOM-Struktur des XOO7-Dokumentes*

kettenwerte und Texte basiert dabei auf Zählern oder der Wiederholung von kurzen Zeichenketten, was die Ergebnisse beim Speichern und Anfragen dieser Werte im Vergleich zu realen Anwendungen verzerren kann.

Das den Daten zugrunde liegende Schema ist fest und regelmäßig, d.h., es gibt weder optionale Attribute oder Elemente noch freien Elementinhalt. Diese Eigenschaft, kombiniert mit der geringen Anzahl unterschiedlicher Elemente, die auch unabhängig von der Dokumentgröße ist, unterstreicht den datenorientierten Charakter des Benchmarks.

Operationen

Da die acht Anfragen des OO7-Benchmarks nur unzureichend das typische Anfragespektrum von XML-Anwendungen abdecken, insbesondere fehlen Navigation und dokumentorientierte Operationen, wurden wichtige Operationstypen ergänzt. Insgesamt umfasst der XOO7-Benchmark 18 Anfragen, die in Tabelle 14-3 aufgeführt sind. Wie schon Xmark kennt auch XOO7 keine Operationen zur Datenmanipulation.

ID	Operation	Kommentar
Q1	Liefere AtomicPart-Elemente, deren MyID mit 5 zufällig erzeugten Werten übereinstimmt.	Einfache Selektion über numerischen Attributwert.
Q2	Liefere jeweils das erste para-Element von 5 Dokumenten, die über den Titel ausgewählt wurden.	Selektion über Zeichenkettenattribut. Elementreihenfolge wird benötigt.
Q3	Selektiere 5% der AtomicPart-Elemente mittels buildDate.	Selektion eines Bereiches über numerischen Attributwert.

Tab. 14-3: *Operationen von XOO7*

ID	Operation	Kommentar
Q4	Liefere AtomicParts-Elemente zusammen mit den zugehörigen Dokumenten.	Verbundoperation über Attributwerte.
Q5	Liefere alle Dokumente, die 2 vorgegebene Phrasen enthalten.	Volltextsuche über Document-Elemente.
Q6	Wiederhole Q1 und ersetze mehrfach auftretende Elemente durch ihre ID.	Testet Fähigkeit zum Erzeugen neuer Ergebnisstrukturen.
Q7	Zähle für jedes BaseAssembly-Element die Anzahl der Dokumente.	Testet Aggregatfunktion.
Q8	Liefere eine absteigend sortierte Liste der CompositePart-Elemente deren buildDate innerhalb eines Jahres vom aktuellen Datum aus liegt.	Testet Sortierung über Attributwerte und den Zugriff auf Umgebungsinformationen.
Q9	Liefere alle BaseAssembly-Elemente eines bestimmten Typs ohne ihre Kindelemente.	Testet Fähigkeit zur Beschränkung der Ergebnismenge auf eine Elementebene.
Q10	Liefere CompositePart-Elemente, deren buildDate-Wert nach dem eines enthaltenen BaseAssembly-Elements liegt.	Testet Effizienz des Zugriffs auf Eltern-Kind-Beziehungen.
Q11	Liefere alle BaseAssembly-Elemente eines Dokumentes, zu denen es BaseAssembly-Elemente gleichen Typs und späteren buildDate in einem anderen Dokument gibt.	Anfrage über mehrere Dokumente mit Vergleichen über Zeichenketten- und numerischen Attributen.
Q12	Liefere alle AtomicPart-Elemente mit ihren CompositePart-Elementen als Kindelemente.	Restrukturierung der Daten durch Vertauschen von Eltern-Kind-Beziehungen.
Q13	Liefere alle ComplexAssembly-Elemente eines bestimmten Typs.	Pfadausdruck mit Wildcards, da nach rekursiv definiertem Element gefragt wird.
Q14	Liefere alle BaseAssembly-Elemente, die nicht von einem gegebenen Typ sind.	Testet Robustheit bei Negation.
Q15	Liefere alle Connection-Elemente innerhalb eines CompositePart-Elementes, deren length-Wert größer als der Durchschnitt ist. Kindelemente sollen nicht mitgeliefert werden.	Gruppierungs- und Durchschnittsfunktion werden benötigt.
Q16	Liefere für ein bestimmtes CompositePart-Element die IDs des Elementes und der zugehörigen Document-Elemente in einem Result-Element zurück.	Testet Fähigkeit zur Erzeugung neuer Ergebnisstrukturen.
Q17	Liefere die Connection-Elemente, die jeweils unter den ersten 5 eines CompositePart-Elementes sind und deren length-Wert über einem vorgegebenen Wert liegt.	Benötigt Informationen zur Elementreihenfolge.
Q18	Liefere für jedes CompositePart-Element die ersten 5 Connection-Elemente, deren length-Wert über einem vorgegebenen Wert liegt.	Ähnlich zu Q18. Hiermit soll getestet werden, ob das Datenbanksystem Optimierungen durchführt.

***Tab 4-3** (Fortsetzung): Operationen von XOO7*

Von den Autoren von XOO7 werden die Operationen in die Gruppen »Traditionelle Datenbankanfragen« (Q1-Q9), »Navigation« (Q10-Q16) und »Dokument-

anfragen« (Q17-Q18) eingeteilt. Allerdings spielen auch in der ersten Gruppe Navigation und dokumentorientierte Aspekte eine Rolle; umgekehrt zeigen die Anfragen der beiden anderen Gruppen ebenfalls Eigenschaften traditioneller Datenbankabfragen.

Metrik

Analog zu Xmark beschränkt auch XOO7 die Testumgebung auf einen einzelnen Computer. Die Daten werden lokal gespeichert und die Anfragen im Einbenutzerbetrieb ausgeführt. Grundlegende Metrik sind demzufolge die Antwortzeiten der einzelnen Anfragen, die unabhängig voneinander ermittelt werden. In [BLLL01] wurden in einer Studie die einzelnen Operationen in die o.g. drei Gruppen aufgeteilt, jeweils 10 mal ausgeführt und pro Gruppe der Mittelwert gebildet. Diese Vermischung von Antwortzeiten unterschiedlicher Anfragen ermöglicht keine aussagekräftige Leistungsbewertung, da es starke Unterschiede zwischen Anfragetypen geben kann und die Durchschnittswerte einseitig von den zeitaufwändigen Anfragen bestimmt werden. Außerdem leidet die Vergleichbarkeit der Ergebnisse aufgrund möglicher Verzerrungen bei wiederholtem Ausführen der Anfragen.

14.7 Vergleich der XML-Benchmarks

In diesem Abschnitt wird ein vergleichender Überblick über die vorgestellten Benchmarks gegeben. Dieser kann als Entscheidungshilfe für die Wahl eines geeigneten Benchmarks genutzt werden. Die hierbei diskutierten Kriterien sind als Übersicht in Tabelle 14-4 zusammengefasst.

Allen vorgestellten Benchmarks ist gemein, dass sie ein möglichst breites Spektrum der XML-Datenverarbeitung abdecken wollen, wodurch sie als Ganzes nur bedingt zur Evaluierung der zu erwartenden Leistungsfähigkeit von reinen dokument- oder datenorientierten Anwendungsszenarios geeignet sind. Trotzdem lassen sich bei den Benchmarks klare Tendenzen zu dem einen oder anderen Typ erkennen. XMach-1 mit seinen verschiedenen Schemata und großem Textanteil bedient schwerpunktmäßig den dokumentorientierten Aspekt, während Xmark und XOO7 hauptsächlich datenorientierte Eigenschaften aufweisen.

Der wesentliche Unterschied liegt jedoch im Skopus der Benchmarks. Nur XMach-1 definiert als Ziel die praxisnahe Evaluierung des gesamten Datenbanksystems im Mehrbenutzerbetrieb. Die beiden anderen Benchmarks beschränken sich auf eine ausführliche Evaluierung des Anfrageprozessors, was sich in der Zahl der Anfragen sowie der Beschränkung auf Einbenutzerbetrieb und lokalen Rechner widerspiegelt.

	XMach-1	Xmark	XOO7
Domäne	Gemischt strukturierte Daten	Gemischt strukturierte Daten	Gemischt strukturierte Daten
Schwerpunkt	dokumentorientiert	datenorientiert	datenorientiert
Skopus	DBMS	Anfrageprozessor	Anfrageprozessor
# Benutzer	Mehrbenutzer	Einbenutzer	Einbenutzer
# Rechner	≥1	1	1
# Schemata	# Dokumente/20	1	1
# Dokumente	10^n ($n \geq 3$)	1	1
Größe der Daten	$16KB*10^n$	10MB-10GB	ca. 4MB-1GB
# Anfragen	8	20	18
# Änderungsoperationen	3	0	0

Tab. 14-4: *Benchmarkvergleich*

Ein weiterer Unterschied besteht in der Aufteilung der Datenbasis in Dokumente. Während XMach-1 die Datenbank auf viele kleinere Dokumente verteilt, erzeugen Xmark und XOO7 jeweils nur ein einziges Dokument, was bei XML-Datenbanken, die ein Dokument als kleinstes Anfragegranulat behandeln, zu Problemen führen kann. Weiterhin kann damit auch nicht der Einfluss verschiedener Schemata getestet werden.

Die Größe der Datenbank selbst lässt sich in allen Benchmarks über Parameter in weiten Grenzen einstellen. Die in der Tabelle angegebenen Obergrenzen resultieren aus den zur Vergleichbarkeit angegebenen Parameterkonfigurationen und nicht aus einer prinzipiellen Beschränkung der Datengeneratoren.

Als einziger der hier vorgestellten Benchmarks unterstützt XMach-1 auch Datenmanipulationsoperationen. Dies wird durch die Spezifikation der Testumgebung unter Einbeziehung eines Applikations-Servers ermöglicht, der einen Ausgleich fehlender oder nicht standardisierter Funktionalität bietet.

Zusammenfassend lässt sich sagen, dass als entscheidendes Kriterium für die Wahl eines der vorgestellten Benchmarks der Skopus betrachtet werden muss. Eine praxisnahe Bewertung des Gesamtsystems ist derzeit nur mit XMach-1 möglich. Sind jedoch Ausführungszeiten und Optimierungsstrategien für einzelne Anfragen das Ziel der Untersuchung, so sind Xmark und XOO7 die erste Wahl. Letztere sind sich in weiten Teilen sehr ähnlich, so dass ein Blick auf die reinen Parameter wenig zur Auswahl beitragen kann. Im Allgemeinen gilt, dass XOO7 mit seinem Rückgriff auf einen objektorientierten Benchmark ein sehr einfaches Datenschema mit wenigen XML-spezifischen Eigenschaften aufweist, so dass in den meisten Fällen Xmark die bessere Wahl darstellen dürfte.

14.8 Erfahrungen und Ergebnisse mit XMach-1

In diesem Abschnitt werden Erfahrungen, die bei der Umsetzung von XMach-1 für XML-Datenbanksysteme gesammelt wurden, sowie Ergebnisse, die einen Eindruck von der derzeitigen Leistungsfähigkeit dieser Systeme vermitteln sollen, vorgestellt.

Die ersten Systeme, für die der Benchmark implementiert wurde, waren native XML-Datenbanken, da diese alle wesentlichen Funktionen, die der Benchmark benötigte, enthielten und somit einen minimalen Anpassungsaufwand versprachen. Dem stand jedoch entgegen, dass diese Datenbanken als Neuentwicklungen noch recht jung am Markt waren und somit Einschränkungen und Fehler aufwiesen, die teilweise eine vollständige Implementierung verhinderten.

Die hauptsächliche Problemquelle bei vielen Systemen lag in der Indexierung – vor allem dem Volltextindex – und dem Mehrbenutzerbetrieb. Im Mehrbenutzerbetrieb waren sowohl Synchronisationsprobleme als auch überproportional gestiegene Antwortzeiten von Leseoperationen während paralleler Schreiboperationen zu verzeichnen. Letzteres lässt sich zum einen auf Sperrgranulate auf Dokumentenebene zurückführen, durch die die Abarbeitung der Anfragen blockiert wird, zum anderen scheinen sich die notwendigen Änderungen an den Indexen beim Einfügen bzw. Löschen von Dokumenten blockierend auf die anderen Anfragen auszuwirken. Bei einer größeren Anzahl von Anfrageprozessen (>20) zeigten einige Systeme Instabilitäten, in deren Folge es zum Abbruch der Anfrageverarbeitung kam. Weitere Problembereiche mit den ersten XML-Datenbankversionen sind in [BöRa01b] aufgeführt.

Operation	NXD 1 alte Version 90%-Antwortzeit (ms)	NXD 1 Optimierung + neue Version 90%-Antwortzeit (ms)
Q1	250	31
Q2	2684	100
Q3	120	11
Q4	80	60
Q5	30	20
Q6	2713	30
Q7	6709	20
Q8	370	11
M1	126069	9173
M2	32306	10114
M3	291	641

Tab. 14-5: *Antwortzeitvergleich für alte und neue Version eines nativen XML-DBS*

Viele dieser Probleme wurden im Laufe von neuen Versionen bereits behoben, was zu einer Steigerung der Anfrageeffizienz führte. Dies verdeutlicht auch Tabelle 14-5. Hier ist jeweils das Maximum der besten 90% der Antwortzeiten zu allen Operationen für ein natives XML-Datenbanksystem gegeben. Die teilweise drastischen Verbesserungen resultieren sowohl aus einer optimierten Implementierung als auch aus dem Einsatz einer neueren Version der Datenbank. Mit dem Versionswechsel wurden einige der Optimierungen erst möglich. Die vergleichsweise langen Antwortzeiten für die Änderungsoperationen M1 und M2 resultieren aus einer wenig effizienten Implementierung des Volltextindex, der bei beiden Operationen aktualisiert werden muss. Die Messungen wurden an einer Datenbank mit 1000 Dokumenten und einem Benutzer auf einem Intel-Pentium-III-Computer (800 MHz) mit einem Speicherausbau von 512 MB durchgeführt.

Ein großes Hemmnis ist jedoch weiterhin die beschränkte Mächtigkeit der Anfragesprachen. Während die nativen XML-Datenbanken wenigstens über vollwertige XPath-Implementierungen und einen direkten DOM-Zugriff verfügen, ist selbst diese Funktionalität bei relationalen Datenbanken mit XML-Erweiterung eingeschränkt und kann oftmals nicht auf Indexstrukturen zurückgreifen.

Die Ausführung von Anfragen über mehrere Schemata hinweg stellt für native XML-Datenbanken inzwischen kaum noch ein Hindernis dar. Ganz anders sieht es hier im relationalen Lager aus. Ein effizienter Zugriff auf die Daten kann hier meistens nur innerhalb eines Schemas erfolgen. Ein Vorteil der langjährigen Entwicklung relationaler Datenbanken wird bei der effizienten Ausführung von Volltextanfragen deutlich, die teilweise den nativen XML-Datenbanken noch Probleme bereiten. Letztere verlagern diese Funktionalität verstärkt auf Fremdprodukte, die über eine Schnittstelle an die Datenbank angebunden werden.

Die Entwicklung der XML-Datenbanken lässt sich auch an der Datenbankgröße, die für eine bestimmte Rohdatengröße benötigt wird, ablesen. In Tabelle 14-6 ist für eine native XML-Datenbank über mehrere Versionen hinweg die Datenbankgröße aufgeteilt nach Daten und Index wiedergegeben. Grundlage dafür bildete die 1000-Dokumente Konfiguration des Benchmarks, die in Rohdatenform ca. 16 Megabyte benötigt. Es ist zu erkennen, dass Daten und Index in der letzten Version weniger als die Hälfte der ursprünglichen Größe belegen. Im Durchschnitt ist für die nativen XML-Datenbanken zu beobachten, dass die Datenbankgröße ohne Index gegenüber den Rohdaten um den Faktor 1,5 bis 3 zunimmt.

	Daten (MB)	Index (MB)
NXD 2 Version 1	64,4	72,4
NXD 2 Version 2	44,5	35,6
NXD 2 Version 3	25,9	31,9

Tab. 14-6: *Änderung der benötigten Datenbankgröße eines nativen XML-DBS*

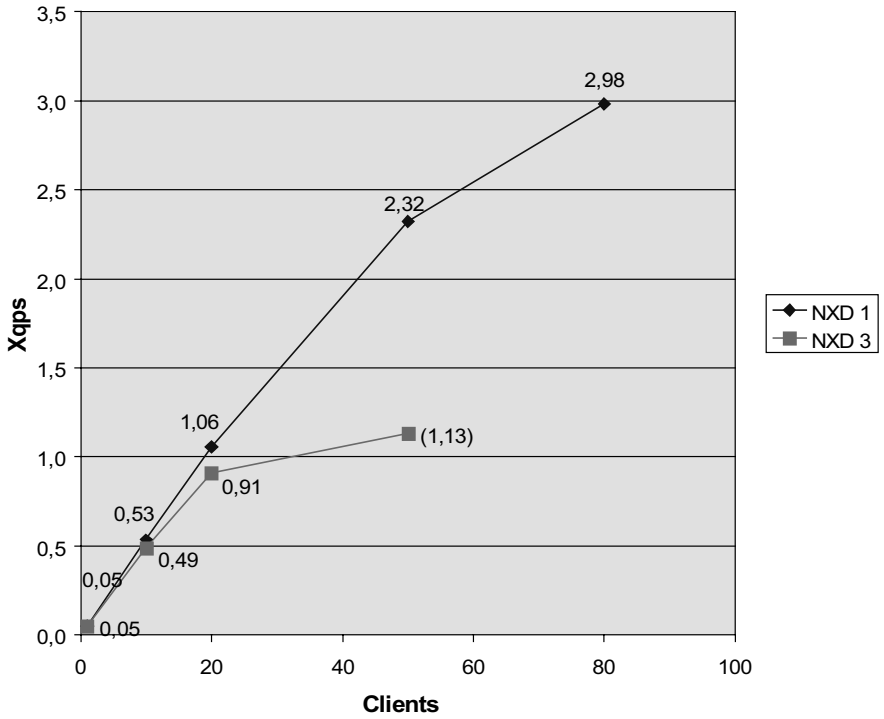


Abb. 14-5: *Vergleich der Durchsatzleistung für zwei native XML-DBS (Juni 2002)*

Das Laden der Datenbank mit 1000 Dokumenten kann je nach System zwischen 90 und 600 s betragen. In Kombination mit der Erstellung der Indexstrukturen reicht die Spanne von 200 bis 800 s für die Population der Datenbank. Bestimmender Faktor ist dabei die Erzeugung des Volltextindex.

Einen Eindruck von der Mehrbenutzer-Leistungsfähigkeit aktueller XML-Datenbanksysteme soll die Grafik in Abb. 14-5 vermitteln. Sie stellt die Durchsatzleistungen zweier nativer XML-DBS in Abhängigkeit von der Anzahl der Clients dar. Die Messungen erfolgten ebenfalls auf dem schon weiter oben erwähnten Intel-Pentium-Computer mit 1000 Dokumenten. Die geringe Datenmenge schließt Behinderungen für den Externspeicherzugriff weitgehend aus, so dass Durchsatzbegrenzungen primär durch CPU- und Sperrengpässe (verursacht durch die Änderungsoperationen) zu erwarten sind. Man erkennt, dass die beiden Systeme NXD 1 und NXD 3 nur einen relativ geringen Durchsatz von ca. 3 bzw. 0,9 Xqps erreichen (beim NXD 3-Wert von 1,13 wurden die Antwortzeitgrenzen überschritten). NXD 1 erzielt eine nahezu lineare Durchsatzsteigerung für bis zu 50-80 Clients, während NXD 3 schon bei 20 Clients seine Leistungsgrenze erreicht. Dieses spiegelt sich ebenfalls in den 90%-Antwortzeiten der einzelnen Operationen wider. Erreicht NXD 3 die vorgegebene 3-Sekunden-Grenze schon bei 20 Clients, ist dieses bei NXD 1 erst bei 80 Clients der Fall. Die im Benchmark nicht restringierten Antwortzeiten der Änderungsoperationen liegen auch im Mehrbenutzerbetrieb meist deutlich höher als für die Queries.

14.9 Zusammenfassung

Benchmarks sind ein wichtiges Instrument zum Vergleich der Leistungsfähigkeit von XML-Datenbanken. Aufgrund des breiten Anwendungsbereiches von XML muss bei der Wahl des Benchmarks die anvisierte Domäne berücksichtigt werden. Die drei vorgestellten Benchmarks berücksichtigen sowohl daten- als auch dokumentorientierte XML-Daten, jedoch mit unterschiedlicher Schwerpunktsetzung. Xmark und XOO7 konzentrieren sich im Wesentlichen auf die Bewertung des Anfrageprozessors im Einbenutzerbetrieb sowie auf eine aus einem XML-Dokument bestehende Datenbank. XMach-1 ist für die Bewertung des Gesamtleistungsverhaltens eines XML-DBS konzipiert und definiert mit Xqps (XML Queries per Second) eine Durchsatzmetrik für den Mehrbenutzerbetrieb. Die Anwendung der Benchmarks auf konkrete Systeme wird durch die derzeit noch geringe Funktionalität der Anfragesprachen erschwert. Die bisher vorliegenden Ergebnisse zeigen, dass sich die Funktionalität und Leistungsmerkmale einzelner Systeme in den letzten beiden Jahren signifikant gebessert haben. Trotzdem bestehen bei einigen XML-Datenbanksystemen immer noch Skalierungsprobleme auf sehr große Datenmengen und Mehrbenutzerfähigkeit.

Literatur

[BLLL01] Bressan, S.; Lee, M. L.; Li, Y. G.; Lacroix, Z.; Nambiar, U.: *The XOO7 XML Management System Benchmark*. National University of Singapore, CS Dept Technical Report TR 21/00, November 2001.

[BöRa01] Böhme, T.; Rahm, E.: *XMach-1: A Benchmark for XML Data Management*. Proc. 9. GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft, S. 264-273, Springer, Berlin, März 2001.

[BöRa01b] Böhme, T.; Rahm, E.: *Benchmarking XML Database Systems – First Experiences,* Position Paper. Ninth International Workshop on High Performance Transaction Systems (HPTS), Pacific Grove, California, Oktober 2001.

[CaDN93] Carey, M. J.; DeWitt, D. J.; Naughton, J. F.: *The OO7 Benchmark*. In Proc. of the ACM SIGMOD Int. Conference on Management of Data, S.12-21, Juni 1993.

[FlKM00] Florescu, D.; Kossmann, D.; Manolescu, I.: *Integrating Keyword Search into XML Query Processing*. In: Proc. of the 9th WWW Conference, Amsterdam, Juni 2000.

[FlKo99] Florescu, D.; Kossmann, D.: *Storing and Querying XML Data using an RDMBS*. In: IEEE Data Engineering Bulletin, Volume 22, Number 3, S. 27-34, September 1999.

[Gray93] Gray, J. (Hrsg.): *The Benchmark Handbook for Database and Transaction Processing Systems*. Morgan Kaufmann Publishers, San Mateo, Kalifornien, 1993.

[RPJA02] Runapongsa, K.; Patel, J. M.; Jagadish, H. V.; Al-Khalifa, S.: *The Michigan Benchmark: Towards XML Query Performance Diagnostics*. http://www.eecs.umich.edu/db/mbench/paper.html, Juni 2002.

[SWKF01] Schmidt, A.; Waas, F.; Kersten, M. L.; Florescu, D.; Manolescu, I.; Carey, M. J.; Busse, R.: *The XML Benchmark Project*. Technical Report INS-R0103, CWI, Amsterdam, Niederlande, April 2001.

[SWKF01b] Schmidt, A.; Waas, F.; Kersten, M. L.; Florescu, D.; Carey, M. J.; Manolescu, I.; Busse, R.: *Why And How To Benchmark XML Databases*. SIGMOD Record, Volume 30, Number 3, S. 27-32, September 2001

Abkürzungen

ADL	Advanced Distributed Learning Initiative
AICC	Aviation Industry Computer-Based Training Committee
API	Application Programming Interface
ASP	Active Server Pages
ASP	Application Service Provider
CAT	Computer Aided Teaching
CBT	Computer Based Training
CDN	Content Delivery Network
CGI	Common Gateway Interface
CLF	Common Logfile Format
CLI	Call Level Interface
COM	Component Object Model
CRM	Customer Relationship Management
DBMS	Datenbank-Management-System (Datenbankverwaltungssystem)
DBS	Datenbanksystem
DCMI	Dublin Core Metadata Initiative
DNS	Domain Name Service
DOM	Document Object Model
DTD	(XML) Document Type Definition
ebXML	Electronic Business using eXtensible Markup Language
EJB	Enterprise Java Beans
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IMS	Instructional Management System Global Learning Consortium
IR	Information Retrieval
JDBC	Java Database Connectivity
JSP	Java Server Pages
JVM	Java Virtual Machine
LMS	Learning Management System

LTSC	IEEE Learning Technology Standards Committee
LOM	Learning Object Metadata
MIME	Multipurpose Internet Mail Extensions
ODBC	Open Database Connectivity
OLAP	Online Analytical Processing
OLE	Object Linking and Embedding
OLTP	Online Transaction Processing
ORDBMS	Objektrelationales *DBMS*
QoS	Quality of Service
RDF	Resource Description Framework
RDBMS	relationales *DBMS*
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SOX	Schema for Object-Oriented XML
SPEC	Standard Performance Evaluation Corporation
SQL	Standard Query Language
SSI	Server Side Include
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TPC	Transaction Processing Performance Council
TTL	Time to Live
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
UUID	Universally Unique Identifier
W3C	World Wide Web Consortium
WBT	Web Based Training
WSDL	Web Services Description Language
WSFL	Web Services Flow Language
WS-Inspection	Web Services Inspection Language
WWW	World Wide Web
XDR	XML-Data Reduced
XML	eXtensible Markup Language
XML-QL	XML Query Language
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformations
XSP	eXtensible Server Pages

Glossar

Im Glossar sind wichtige Begriffe zu den Themen des Buchs zusammengestellt. *Kursive* Bezeichnungen beziehen sich auf Querverweise innerhalb des Glossars.

5-Minuten-Regel
Faustregel für die Konfiguration einer *Cache*-Größe aufgrund des Verhältnisses von Speicherkosten und Zugriffsdurchsatz.

Änderungslokalität
Ermöglicht eine isolierte Modifikation der verschiedenen Aspekte eines *Web-Informationssystems* bestehend aus Inhaltsaspekt (die eigentlichen Daten), Hypertextaspekt (struktureller Aufbau einer Webseite inklusive Navigation) und Präsentationsaspekt (grafische Repräsentation und Layout).

Anfrageoptimierung
Prozess der Transformation einer internen Anfragedarstellung in einen bezüglich eines Kostenmodells möglichst optimal ausführbaren Plan.

API (Application Programming Interface)
Menge mit Funktionsaufrufen, die für Anwendungen zur Verfügung gestellt wird, z.B. *ODBC* und *JDBC* im Datenbankbereich, ISAPI und NSAPI für *HTTP-Server* oder *DOM* und *SAX* für den Zugriff auf *XML-Daten*.

Applet
Am *Web-Server* abgelegte Java-Programme, die bei Aufruf dynamisch – in ein HTML-Programm eingebettet – zum Web-*Client* transferiert und dort durch die JVM (Java Virtual Machine) ausgeführt werden.

Applikations-Server
Ein Server, der für das Abarbeiten der Anwendungslogik einer Web-Applikation zuständig ist, dadurch den *Web-Server* entlastet und eine bessere Skalierbarkeit ermöglicht. Der Begriff wird auch für die zur Verwaltung und Ablaufkontrolle der Anwendungen verantwortliche System-Software verwendet.

ASP (Active Server Pages)
Realisierungsform für *dynamische HTML-Seiten*, bei der in Webseiten eingefügte Skript-Anweisungen serverseitig ausgewertet werden. Einsatz in Microsoft-Server-Umgebungen; in anderen Umgebungen steht mit *JSP* eine ähnliche Funktionalität zur Verfügung.

ASP (Application Service Provider)
Anbieter für Software-Dienstleistungen, welche auf einer bereitgestellten IT-Infrastruktur gegen eine Nutzungsgebühr erbracht werden.

Assoziationsregeln
Ergebnis eines speziellen *Data-Mining*-Verfahrens; drücken einen statistisch signifikanten Zusammenhang zwischen z.B. Kunden und von diesen gekauften Dingen aus (Warenkorbanalyse).

Autorensystem
System zur Erstellung und zum Austausch von Lerninhalten.

B- / B*-Baum
Klassische *Indexstruktur* im Datenbankbereich, unterstützt *Werteanfragen*.

Benchmark
Verfahren zur Leistungsmessung eines Systems oder einzelner Komponenten.

Browser
Client-Programm zur Darstellung von HTML-Dateien und weiteren Web-Inhalten; ermöglicht insbesondere komfortable Fortbewegung im Web.

Cache
Zusätzlicher Speicher, in dem Daten zum schnelleren Zugriff vorgehalten bzw. dynamisch repliziert werden. Damit sollen insbesondere langsame Datenzugriffe auf Externspeicher und entfernte Rechner eingespart werden.

Cache-Ersetzungsstrategie
Verfahren zur Auswahl eines oder mehrerer Datenobjekte, die aus dem *Cache* entfernt werden, um Platz für ein neues Objekt zu schaffen.

Cache-Hierarchie
Folge bzw. Hierarchie von *Caches*, z.B. entlang des Pfades von einem *Client* zu einem *Web-Server* bzw. dem permanenten Speicherort von Daten.

Cache-Kohärenzsteuerung
Verfahren zur Invalidierung veralteter Kopien von Datenobjekten in einem *Cache* und zur Aktualisierung der Kopien.

Caching
Methode zum dynamischen Replizieren von Daten zum Zweck des schnelleren Zugriffs (*hierarchisches Caching*, *kooperatives Caching*, *nutzenorientiertes Caching*).

Caching, hierarchisches
Methode zur Nutzung einer *Cache-Hierarchie*.

Caching, kooperatives
Caching-Methode, bei der sich mehrere Rechner derselben *Caching-Stufe* ihre replizierten Daten auf Anfrage gegenseitig zur Verfügung stellen oder auf Wunsch eines anderen Rechners an *Clients* weiterleiten.

Caching, nutzenorientiertes
Cache-Ersetzungsstrategien, die Datenobjekte aufgrund einer Nutzenmetrik ersetzen, welche die Zugriffshäufigkeit, die Größe und die Zugriffskosten der Datenobjekte berücksichtigt. Ein Beispiel ist das sog. GreedyDual-Size-Verfahren (siehe Kap. 7).

Caching-Stufe
Cache einer bestimmten Stufe in einer *Cache-Hierarchie*.

Caching, temperaturorientiertes
Cache-Ersetzungsstrategien, welche Datenobjekte aufgrund ihrer *Temperatur* ersetzen.

CGI (Common Gateway Interface)
Schnittstelle für die Kommunikation eines *HTTP-Servers* mit externen Programmen zur Generierung *dynamischer HTML-Seiten*.

Clickstream-Analyse
Auswertung von *Web-Log*-Daten zur Bewertung des Web-Nutzungsverhaltens.

Click-through-Rate
Anteil der Besucher einer Webseite, die einen weiterführenden Link anklicken (z.B. Verhältnis der Klicks auf eine Produktseite zu den Klicks auf das referenzierende Werbe-Banner).

Client
Rechner oder elektronisches Gerät (z.B. Handy) eines Endbenutzers bzw. Rechner/Programm, von dem aus Programme (Dienste) eines Servers aufgerufen werden.

Clustering
Data-Mining-Verfahren, bei welchem versucht wird, in einer statistisch relevanten Sammlung von Exemplaren Häufungspunkte bzgl. bestimmter Eigenschaften zu identifizieren.

Computer Aided Teaching: siehe *Computer-unterstützte Lehre*

Computer Based Training
Eine Form von Training, bei der ein Computer eingesetzt wird.

Computer-unterstützte Lehre
Die (meist universitäre) Lehre unter Einsatz des Computers, wobei die Interpretationen von der Slide-Show auf einem Laptop über den mit Rechnern gefüllten Klassenraum bis zum Electronic White Board reichen.

Content-Delivery-Netze
Ins Internet eingebettete Subnetze mit intensivem *Caching* auf *Edge-Server*-Rechnern, von denen aus große Datenmengen sehr effizient an Clients transferiert werden können.

Cookie
Kleine Textdatei, die vom *HTTP-Server* zum *Browser* gesendet und dort zu Informationszwecken gespeichert wird.

Data Guide
Index aus dem Lore-System zur Indexierung von Pfaden.

Data Mining
Sammlung von Verfahren zur Entdeckung von Mustern in großen Datenmengen. Wesentliche Verfahrensklassen sind u.a. *Clustering*, *Klassifikation* und die Bestimmung von *Assoziationsregeln*.

Data Warehouse
Datenbank, die Daten aus unterschiedlichen Quellen integriert und Analysen darauf unterstützt.

Datenbanksystem
System zur dauerhaften, anwendungsunabhängigen und strukturierten Speicherung großer Datenbestände basierend auf einem *Datenmodell*. Es stellt seinen Benutzern eine Sprache wie *SQL* zur Datendefinition und -manipulation und Anfragen zur Verfügung. Intern unterstützt es u.a. *Anfrageoptimierung*, Transaktionsverwaltung und Externspeicher-Verwaltung. Je nach *Datenmodell* spricht man von relationalen, objektorientierten und objektrelationalen Datenbanksystemen. Siehe auch *XML-Datenbanksystem*.

Datenmodell
Abstraktionsmechanismus zur Beschreibung der konzeptionellen Aspekte (Entitäten und deren Beziehungen) einer Datenbank sowie der anwendbaren Operatoren. Die größte Bedeutung hat das relationale Datenmodell, wohingegen objektorientierte Datenmodelle nur relativ geringe Verbreitung erreichen konnten. Zunehmend an Bedeutung gewinnen objektrelationale Datenmodelle, bei denen das Relationenmodell um objektorientierte Konzepte erweitert wird.

Datenmodell, semistrukturiertes
Datenmodell, welches eine angemessene Beschreibung von Daten, deren Struktur variabel ist, ermöglicht. Basiert typischerweise auf einem Graphmodell und unterstützt pfadorientierte Zugriffe auf die Inhalte der Graphknoten. Kann in rechnerinterner (*OEM*) und in textbasierter Repräsentation (*XML*) vorliegen.

Dienst siehe *Web Service*

Dienstkomposition
Entwicklung von zusammengesetzten Diensten, d.h. *Diensten*, die andere Dienste ansprechen und ihre Ergebnisse verwenden.

Distance Learning
Lehren und Lernen über eine räumliche Entfernung hinweg, also z.B. vom häuslichen Arbeitsplatz bzw. von der Wohnung aus oder im Rahmen von mobilen Rechnern. Auch *Tele Teaching* genannt.

Dokumenttyp-Definition siehe *DTD*

Dokumenttyp-Deklaration
Optionale Anweisung im *Prolog* eines *XML-Dokumentes*, die dem Dokument eine *DTD* zuordnet. Es kann sowohl auf eine externe DTD verwiesen werden, als auch eine DTD innerhalb der Deklaration angegeben werden.

DOM (Document Object Model)
Sprach- und plattformunabhängiges Modell zur Überführung des zeichenorientierten *XML*-Modells über einen Parserbaum in ein rechnerinternes (objektorientiertes) Datenmodell. Implementiert als *XML-Prozessor*, bietet DOM gleichzeitig eine Schnittstelle zur Anwendungsprogrammierung an (*API*), mit deren Hilfe Anwendungen navigierend auf den Graph eines XML-Dokuments und auf die Knoteninhalte der Graphknoten zugreifen können. DOM wird als pfadorientiert bezeichnet und unterliegt den Standardisierungsbemühungen des *W3C*.

DTD (Document Type Definition)
Grammatik zur Definition einer Klasse von *XML-Dokumenten*. Insbesondere beschreibt sie die Strukturierung der Elemente in Form von *Elementtyp-Deklarationen*. DTDs dienen einem *XML-Prozessor* zur Feststellung, ob ein XML-Dokument *gültig* ist. Die einem XML-Dokument zugehörige DTD wird durch die *Dokumenttyp-Deklaration* festgelegt.

dynamische HTML-Seiten
HTML-Seiten, die mittels Datenbankzugriffen und Applikationsprogrammen dynamisch (zur Zugriffs- bzw. Anforderungszeit) auf einem Web-Server zusammengestellt werden.

E-Business
Abwicklung von Geschäftsabläufen über das Internet oder andere Computernetzwerke.

ebXML
Modulare Zusammenstellung XML-basierter Standards im Bereich *E-Business*.

Edge-Server
Rechner, auf dem Daten dynamisch repliziert sind (siehe auch *Caching*) und der bzgl. der Zugriffsgeschwindigkeit möglichst nahe bei einer großen Gruppe von *Clients* platziert ist.

Element Content (Elementinhalt)
Spezielle Ausprägung einer *Elementtyp-Deklaration* in einer *DTD*. Man spricht von Elementinhalt, wenn ein Elementtyp nur Kindelemente, jedoch keine Zeichendaten enthält. Die Strukturierung des Elementinhalts wird auch als Inhaltsmodell (content model) bezeichnet.

Elementtyp-Deklaration
Definition eines Elementtyps in einer *DTD*. Ein Elementtyp besteht aus dem Elementnamen, den zugehörigen Attributen und einer Inhaltsdefinition (*element content*, *mixed content*).

EJB (Enterprise Java Beans)
Server-basiertes Rahmenwerk, das die Erzeugung von portierbaren, auf entsprechenden Servern vorgehaltenen und in der Programmiersprache Java implementierten Komponenten erlaubt.

Electronic Business siehe *E-Business*

Endpunkt
Eine Adresse in einem Netzwerk, unter der ein Dienst erreichbar ist (im Kontext von *WSDL*).

Entity
Physisches Strukturkonzept von *XML*, welches Sonderzeichen sowie interne und externe in sich abgeschlossene Speichereinheiten bezeichnet. Entities realisieren das Konzept der Modularisierung in *XML*.

False Drop
Datenobjekt, das über einen *Index* (z.B. basierend auf *Signaturen*) ermittelt wird, ohne jedoch zur Antwortmenge einer Anfrage zu gehören.

FTP (File Transfer Protocol)
Protokoll zur Übertragung von Dateien zwischen Rechnern.

Get-If-Modified-Since
Konditionaler *HTTP*-Zugriff bzw. Aktualitätstest für ein Datenobjekt im Web.

Gültig
Ein *XML-Dokument* ist gültig (valide), wenn es eine dazugehörige *DTD* besitzt und das Dokument die darin formulierten Beschränkungen einhält (siehe auch *schemagültig*).

Heterogenität
Bei der *Integration* von mehreren Datenquellen im Web muss deren Heterogenität überwunden werden. Heterogenität bezeichnet die Verschiedenheit in der Speicherung von Daten(repräsentationen) desselben Anwendungsgebietes, etwa durch deren Darstellung in unterschiedlichen Datenmodellen oder unterschiedlicher Modellierung. Siehe auch *Integrationskonflikt*.

Hidden Fields
Erlauben das Zwischenspeichern von Zustandsinformationen in versteckten Formularfeldern in einer *URL*.

Hit
Zugriff auf eine *Website*-Datei.

Hitze
Maß für die Zugriffshäufigkeit eines Datenobjekts.

HTTP (Hypertext Transfer Protocol)
Kommunikationsprotokoll des Web, v.a. zwischen *Browser* und *HTTP-Server.*

HTTPS (Hypertext Transfer Protocol Secure)
Kombination des *HTTP*-Protokolls mit *SSL,* um eine verschlüsselte Übertragung von Daten zu ermöglichen.

HTTP-Server
Rechner, der *HTTP*-Zugriffe bedient und ggf. Applikationsprogramme zur Konstruktion *dynamischer HTML-Seiten* aufruft. Oft auch als *Web-Server* bezeichnet.

IDREF
Ein *XML*-Attributtyp zur Realisierung von Verweisen zwischen Elementen innerhalb eines XML-Dokumentes. Jeder Verweis bezieht sich auf ein Attribut vom Typ ID, das zur eindeutigen Identifizierung von Elementen dient.

Information Retrieval
Sammlung von Methoden zur inhaltlichen Suche in Dokumenten.

Index(struktur)
Datenstruktur, die schnellen Zugriff auf für eine Anfrage relevante Daten erlaubt.

Inhaltsmodell siehe *element content*

Integration
Integration bezeichnet den Prozess der Zusammenführung heterogener Datenquellen in eine einheitliche Repräsentation. Neben der Transformation in das Zieldatenmodell werden Heterogenitäts- bzw. *Integrationskonflikte* gelöst, etwa unterschiedliche Modellierungsentscheidungen oder Detaillierungsgrade. Je nach betrachteter Ebene unterscheidet man Systemintegration, Anwendungsintegration, Schemaintegration und Datenintegration. Diese stellen Teilaspekte einer umfassenden Integration dar.

Integrationskonflikt
Die Ursachen von Konflikten bei der Integration liegen auf drei verschiedenen Ebenen: Datenmodellheterogenität, Schemaheterogenität (oder heterogene Modellierung) und Heterogenität auf der Datenebene. Je nach Ebene sind unterschiedliche Probleme, genannt Konflikte, zu behandeln.

Invertierte Liste
Einfache *Indexstruktur* aus dem Bereich *Information Retrieval* bzw. aus Datenbanksystemen.

JDBC (Java DataBase Connectivity)
Bekanntester und am weitesten ausgereifter Ansatz zum dynamischen Zugriff auf relationale *Datenbanksysteme* von Java-Programmen aus.

JSP (Java Server Pages)
Von Sun Microsystems eingeführte Erweiterung des *Servlet*-Konzepts zur Entwicklung portabler Web-Anwendungen mit dynamisch generierten Ergebnissen. Java-basiertes Pendant zu *ASP (Active Server Pages)*.

Klassifikation
Data-Mining-Verfahren, bei welchem eine statistisch relevante Sammlung von Exemplaren nach vorgegebenen Eigenschaften unterteilt wird.

Kostenmodell
Modell zur Bestimmung der erwarteten Ausführungskosten von Datenbankoperationen; in der Regel basierend auf statistischen Informationen über die gespeicherten Daten.

Learning Management System
System zur Integration und zur Speicherung von Inhalten, die eventuell von unterschiedlichen Autoren erstellt wurden, sowie zur Verwaltung von Informationen über Inhalte und über Lernende.

Lernobjekt
Abgeschlossene Einheit digitalisierten Lerninhalts, welche potenziell in verschiedenen Kontexten (wieder) verwendet werden kann.

LFU-Strategie (Least Frequently Used)
Cache-Ersetzungsstrategie, bei der das Datenobjekt mit der niedrigsten Zugriffshäufigkeit ersetzt wird.

LRU-Strategie (Least Recently Used)
Cache-Ersetzungsstrategie, bei der das Datenobjekt ersetzt wird, dessen letzter Zugriff am weitesten in der Vergangenheit liegt.

LRU-k-Strategie
Verallgemeinerung der *Cache-Ersetzungsstrategie LRU*, bei der das Datenobjekt ersetzt wird, dessen k-letzter Zugriff am weitesten in der Vergangenheit liegt.

LRFU
Cache-Ersetzungsstrategie, die *LRU* und *LFU* kombiniert.

Markov-Kette
Mathematisches Modell, mit dem u.a. Zugriffswahrscheinlichkeiten auf Datenobjekte für Prefetching- und Caching-Zwecke geschätzt werden können.

Mediator
Softwarekomponente zur Vereinfachung, Reduzierung, Kombination und Erklärung von Daten. Wird v.a. zur Bereitstellung einer gemeinsamen Anfragemöglichkeit auf unterschiedliche Datenquellen genutzt.

Metadaten
Daten zur Beschreibung von Daten.

Metrik
Maß, in dem Werte (z.B. Ergebnisse eines *Benchmarks*) angegeben werden.

MIME (Multipurpose Internet Mail Extensions)
Ein Standardformat für E-Mail. Mit Hilfe verschiedener MIME-Typen ist es möglich, Audio-, Video-, Bild- oder HTML-Dateien zu versenden.

Mixed Content (Gemischter Inhalt)
Spezielle Ausprägung einer *Elementtyp-Deklaration* in einer *DTD*. Man spricht von gemischtem Inhalt, wenn ein Elementtyp einfachen Text bzw. einfachen Text in Verbindung mit weiteren Kindelementen enthalten darf. Die Reihenfolge der Kindelemente kann hierbei nicht festgelegt werden.

Model-View-Controller-Prinzip
Grundprinzip im Software-Engineering, das die Trennung der Aspekte Inhalt (Model), Steuerungskomponente (Controller) und Präsentation (View) fordert.

Namensraum
Zusammenstellung von Namen, identifiziert durch einen URI-Verweis, die in XML-Dokumenten als Elementtypen und Attributnamen verwendet werden können. Zur eindeutigen Qualifizierung wird bei der Verwendung der Namen ein Präfix mit dem Bezeichner des Namensraums und ein Doppelpunkt vorangestellt.

Natives XML-Datenbanksystem siehe *XML-Datenbanksystem, natives*

Navigierende Anfrage
Anfrage, die sich auf die Struktur eines Dokuments bezieht (siehe auch *Pfadausdruck*).

OEM (Object Exchange Modell)
Rechnerinternes, semistrukturiertes Datenmodell; Grundlage für eine Anzahl von Projekten, die sich mit der programmiertechnischen Darstellung semistrukturierter Daten befassen.

Page View
Vollständig angezeigte Seite einer *Website* (inklusive eingebetteter Komponenten wie Bilder, Navigationsmenüs etc.).

Parser
Software-Modul, welches eine lexikalische Analyse von Programmtext (im Compilerbau), Anfragen (bei Datenbanken) und/oder Dokumenten (im Falle von XML) durchführt und die Einhaltung einer vorgegebenen Spezifikation prüft. Ein *XML-Prozessor* enthält einen Parser für XML-Dokumente.

Peer-to-Peer-Anwendungen
Anwendungen, bei denen nicht mehr zwischen Client- und Server-Rechnern unterschieden wird, sondern alle Rechner gleichberechtigte Partner (Peers), z.B. zum Internet-weiten File-Sharing, sind.

Pfad(ausdruck)
Sequenz von Namen zur Navigation innerhalb einer baumartigen bzw. graphbasierten Datenstruktur. In *XML-Dokumenten* können mit Pfadausdrücken bestimmte Elemente oder Attribute ausgewählt werden.

Pfadindex
Index, der strukturelle Suche in (XML-)Dokumenten unterstützt (typische Vertreter sind z.B. *Data Guides*, T-Indexe).

Portal
Hier: Aggregation der durch einen Portlet-Server angebotenen *Portlets*.

Portlet
(Java-)Programm, welches innerhalb eines (oder mehrerer) *Portale* eingebunden ist, innerhalb einer Portal-Umgebung ausgeführt wird und spezifische Dienste umsetzt. Portlets können vom Benutzer zum Zweck der Erstellung einer individuellen *Website* ausgewählt, angepasst und aggregiert werden.

Prefetching
Vorladen eines Datenobjekts in einen *Cache* zum Zweck der Maskierung der Zugriffslatenz.

Prefetching, nutzenorientiertes
Strategie zum Vorladen von Datenobjekten aufgrund einer Nutzenmetrik, welche die Zugriffshäufigkeit, die Größe und die Zugriffskosten von Kandidatenobjekten und den im Cache gehaltenen Objekten berücksichtigt.

Prefetching, temperaturorientiertes
Strategie zum Vorladen von Datenobjekten aufgrund der *Temperatur* von Kandidatenobjekten und der im *Cache* gehaltenen Objekte.

Prolog
Beginn eines *XML-Dokumentes* bestehend aus XML-Deklaration und *Dokumenttyp-Deklaration* als Informationen für den *XML-Prozessor*. Beide Deklarationen sind optional, jedoch wird empfohlen, die XML-Deklaration anzugeben. An den Prolog schließt sich das Wurzelelement an.

Proxy
Übernimmt die Kommunikation zwischen einem Benutzer und einem *Web Service* (im Kontext *Web Service*).

Proxy-Server
Rechner, über den eine Gruppe von Clients mit dem Internet verbunden ist.

Publish-Subscribe-Architektur
Kombination aus *pull-basierter* und *push-basierter Architektur*, bei der Clients Daten abonnieren können, die bei Änderungen vom Server unaufgefordert zu den Clients geschickt werden.

Pull-basierte Architektur
Architektur, bei der Server nur dann geänderte Daten an Clients schicken, wenn sie von den Clients explizit angefordert werden.

Push-basierte Architektur
Architektur, bei der Server geänderte Daten unaufgefordert an Clients schicken.

Quality-of-Service
Güte eines Informationsdienstes, die idealerweise garantierte Zugriffszeiten umfasst. Werden je nach Kunden-, Verbindungs- und Applikationsart unterschiedliche Gütegarantien gegeben, spricht man von »differentiated Quality-of-Service«.

Referrer
URL und damit *Website*, von der auf eine andere URL derselben oder einer anderen Website verwiesen wird.

Reverse-Proxy
Rechner, über den alle eingehenden Zugriffswünsche an eine Web-Server-Farm laufen.

RPC (Remote Procedure Call)
Aufruf von Funktionen über Rechnergrenzen hinweg.

SAX (Simple API for XML)
Spezieller *XML-Prozessor*, der neben den Aufgaben der Durchmusterung und Verifikation von *XML-Dokumenten* gleichzeitig ereignisgesteuert Anwendungen den Zugriff auf die Knoten und Knoteninhalte eines XML-Baumes ermöglicht. Im Unterschied zu *DOM* wird der Parserbaum bei SAX nur temporär aufgebaut. SAX unterliegt nicht den Standardisierungsbemühungen des *W3C*.

Schema
Beschreibung einer Datenstruktur inklusive Typinformationen.

Schemagültig
Als schemagültig wird ein *XML-Dokument* bezeichnet, wenn (anstelle einer *DTD)* ein *XML-Schema* existiert, gegen das das Dokument validiert werden kann.

Schema Matching
Klasse von Verfahren zur (möglichst automatisierten) Bestimmung der Übereinstimmungen zwischen mehreren *Schemata*. Wesentlicher Teilschritt bei der Daten- bzw. Schemaintegration oder bei der Abbildung von XML-Nachrichten in E-Business-Anwendungen.

Service siehe *Web Service*

Servlet
Programm einer Web-Applikation, das auf einem *Web-Server* läuft und typischerweise in einer Skriptsprache oder in Java geschrieben ist. Servlets können verwen-

det werden, um *dynamische HTML-Seiten* zu erzeugen und/oder Datenbankzugriffe zu regeln.

Semistrukturiertes Datenmodell siehe *Datenmodell, semistrukturiertes*

SGML (Structured Generalized Markup Language)
Standardisierte, umfangreiche Auszeichnungssprache (markup language) zur Strukturierung von Dokumenten. Ursprung der Sprachen HTML und *XML*.

Signatur
Bitvektor, auf den mittels einer Hash-Funktion Werte abgebildet werden.

Skalierung
Änderung von Größen relativ zu ihren Ausgangswerten (z.B. Vergrößerung einer Datenbank um einen bestimmten Faktor).

Skript
Anweisungsfolge in einer Skriptsprache.

SMTP (Simple Mail Transfer Protocol)
Protokoll, das verwendet wird, um Nachrichten (E-Mails) an andere Rechner zu verschicken.

SOAP (Simple Object Access Protocol)
Ein auf *XML* basierendes Protokoll für den Datenaustausch, das unter anderem *RPC*-Aufrufe ermöglicht. Die Standardisierung erfolgt durch das *W3C*.

SOAP-Dokument .
XML-Dokument, das eine *SOAP*-Nachricht gemäß dem *SOAP*-Protokoll enthält.

Speicherung von XML-Dokumenten
XML-Dokumente können unstrukturiert als Ganzes oder zerlegt (strukturiert) in Datei- oder *Datenbanksystemen* gespeichert werden. Bei der zerlegten Speicherung (Dekomposition) erfolgt entweder eine anwendungsunabhängige, generische Speicherung der Dokumente, typischerweise im Rahmen eines Graphenmodells. Alternativ dazu erfolgt eine Abbildung auf ein anwendungsbezogenes Schema von (objekt-)relationalen oder objektorientierten Datenbanksystemen. Daneben existieren hybride Speicherungsformen (siehe Kap. 5).

SQL (Structured Query Language)
Standardisierte und wichtigste Anfragesprache für Datenbanken, die praktisch von allen relationalen *Datenbanksystemen* unterstützt wird. Wichtige Versionen sind SQL-92 und SQL:1999.

SQLJ (SQL Java)
Ein von führenden Datenbankherstellern entwickelter Standard, der es erlaubt, *SQL* statisch in Java-Programme einzubetten.

SQLX (SQL/XML)
Vorschlag zur Erweiterung des *SQL*-Standards um einen Datentyp XML, um damit *XML-Dokumente* innerhalb relationaler bzw. objektrelationaler *Datenbanksysteme* speichern und verarbeiten zu können.

SSI (Server Side Include)
Technik zur Erweiterung von Webseiten um bestimmte (SSI-) Befehle, welche durch den *Web-Server* vor dem Versenden ausgeführt werden.

SSL (Secure Socket Layer)
Verfahren zur sicheren Übertragung verschlüsselter Daten, das in Kombination mit HTTP, *FTP*, und anderen Kommunikationsprotokollen einsetzbar ist.

Teleteaching siehe *Distance Learning*

Temperatur
Pro Speicherplatzeinheit normalisiertes Maß für die Zugriffshäufigkeit von Datenobjekten.

Time-to-Live
Schätzwert für die Zeitspanne, während der ein Datenobjekt nicht geändert wird; als Grundlage für die Invalidierung und Aktualisierung von Kopien in einem *Cache*.

Trie
Ein (binärer) Suchbaum, in dem die Kanten mit Zeichen beschriftet sind.

UDDI (Universal Description, Discovery and Integration)
Verzeichnisdienst und -standard für *Dienst*-Metadaten, insbesondere Daten über Dienstanbieter, Dienstgruppen und technische Spezifikationen.

UDDI-Wolke
Server-Verbund zur Realisierung eines globalen *UDDI*-Verzeichnisdienstes.

URC (Universal Resource Citation)
Schlägt die Brücke zwischen *URL* und *URN*. Die URC verweist in eine Tabelle, in welcher zum logischen Namen die physische Adresse festgehalten wird.

URI (Uniform Resource Identifier)
Abstrakter Oberbegriff für eineindeutige Identifikatoren von Web-Ressourcen in Form von *URL*, *URN* und *URC*.

URL (Uniform Resource Locator)
Gibt den physischen Aufenthaltsort einer Ressource an, etwa den Speicherplatz einer HTML-Seite (z.B. http://www.cs.uni-essen.de/DAWIS).

URN (Uniform Resource Name)
Eineindeutige Zeichenkette (Name) für eine beliebige Ressource im Sinne einer Benennung, aber keine direkte Speicherortangabe. Diese Namespace ID genannte

Zeichenkette unterliegt einem globalen Registrierungszwang, um ihre Eindeutigkeit zu gewährleisten.

UUID (Universally Unique Identifier)
Weltweit eindeutiger Identifikator für *Dienst*-Metadaten-Dokumente (im Kontext von *UDDI*).

Valide siehe *gültig*

Volltextanfrage
Anfrage, die inhaltlich auf Textdokumenten sucht.

Volltextindex
Index, der inhaltliche Suche in Textdokumenten unterstützt (typische Vertreter sind z.B. *invertierte Listen*, *Tries*).

W3C (World Wide Web Consortium)
1994 gegründetes Konsortium zur Entwicklung von Standards, Spezifikationen, Protokollen, Programmen und Werkzeugen für das Internet. Mittlerweile umfasst es mehr als 510 Mitglieder und ist eines der wichtigsten Standardisierungsgremien im Bereich der Informationstechnik. Für weitere Information siehe http://www.w3c.org/Consortium/ .

W3C Recommendation
Vom *W3C* verabschiedete Spezifikation, die den höchstmöglichen Status einer Empfehlung (Recommendation) besitzt.

W3C-Standard
Umgangssprachlicher Terminus für *W3C Recommendation*. Das *W3C* bezeichnet seine Spezifikationen jedoch nicht als »Standard«.

Web Based Training
Lernen, welches als Kommunikationsmedium das Internet verwendet und auf der Seite eines Teilnehmers im Allgemeinen nicht mehr als einen Browser erfordert.

Web-Informationssystem
Im Web zugängliches Informationssystem.

Web-Informationssystem, statisches
Web-Informationssystem, bei dem (fertige) am Web-Server vorliegende HTML-Dateien ausgegeben werden, im Gegensatz zu dynamischen, zur Anfragezeit des Clients erzeugten Webseiten.

Web-Informationssystem, ubiquitäres
Web-Informationssystem, das personalisierte Dienste zu jeder Zeit, an jedem Ort und über beliebige Medien, also allgegenwärtig zur Verfügung stellt.

Web-Log
Vom *Web-Server* geführte Dateien, in der sämtliche *Website*-Zugriffe chronologisch protokolliert sind.

Web-Server
Rechner oder Gruppe von Rechnern, die zusammen eine Web-Anwendung realisieren, also Oberbegriff für HTTP-, Applikations-, Daten-Server, aber auch als Synonym für *HTTP-Server* verwendet.

Web-Server-Accelerator siehe *Reverse-Proxy*

Web-Server-API
Von verschiedenen *HTTP-Server*-Herstellern jeweils angebotene proprietäre Server-Erweiterung (ansonsten ähnlich zu *CGI*).

Web-Server-Farm
Gruppe von gleichartigen Web-Server-Rechnern, über die die Last einer Web-Anwendung verteilt wird.

Web Service
Bisher keine einheitliche Definition des Begriffs. Im üblichen Sprachgebrauch bezeichnet ein Web Service einen Dienst, der Benutzern über das Web zur Verfügung gestellt wird und dabei beispielsweise auf *XML* und *HTTP* zurückgreift und auf automatisierte Benutzung ausgerichtet ist.

Web Service Engine
Anwendung, die es erlaubt, *Web Services* im Internet verfügbar zu machen.

Website
Menge der Webseiten, die von einem *Web-Server* bzw. von einer Menge zusammengehöriger Web-Server eines Unternehmens bzw. einer Einrichtung verwaltet werden.

Web Usage Mining
Web-Zugriffsanalyse unter Anwendung von *Data-Mining*-Verfahren.

Web-Zugriffsanalyse
Analyse des Web-Nutzungsverhaltens, typischerweise bezogen auf eine *Website*.

Werteanfrage
Anfrage, die nach exakten Werten oder Wertebereichen in Datenobjekten sucht.

Wohlgeformt
Ein *XML-Dokument* heißt wohlgeformt, wenn es den XML-Syntaxregeln genügt (well-formed gemäß dem *W3C-Standard* XML 1.0).

Wrapper
Softwarekomponente, die den Inhalt einer Datenquelle zur Vereinheitlichung in einem anderen Datenmodell oder Schema repräsentiert; etwa ein XML-Wrapper für eine relationale Datenbank.

WSDL (Web Services Description Language)
Sprache zur Beschreibung der Schnittstellen von *Web Services* (Vorschlag für einen *W3C-Standard*).

WSFL (Web Services Flow Language)
Sprache zur Beschreibung von zusammengesetzten *Diensten* (*Dienstkomposition*).

WS-Inspection (Web Services Inspection Language)
Spezifikation zur Bereitstellung von Verweisen auf Informationen über *Web Services* auf dem *Web-Server*.

WS-Inspection-Dokument
XML-Dokument, das Verweise auf Informationen über *Web Services* enthält und dem *WS-Inspection*-Standard genügt.

XLANG
Sprache zur Beschreibung von zusammengesetzten *Diensten* (*Dienstkomposition*). Erweiterung von *WSDL*.

XML (eXtensible Markup Language)
Textbasiertes, *semistrukturiertes Datenmodell*; Untermenge von *SGML*. Durch die Textform ist XML für Menschen lesbar. Zur rechnerinternen Darstellung und Verarbeitung von XML-Daten bedarf es weiterer Programme bzw. Schnittstellen (z.B. *DOM* oder *SAX*). Seit 1996 in der Entwicklung durch das *W3C*; 1998 Verabschiedung der *W3C Recommendation* XML 1.0.

XML-Daten
Menge von *XML-Dokumenten* bzw. XML-Datenobjekten.

XML-Datenbanksystem
Datenbanksystem zur Verwaltung von *XML-Dokumenten*.

XML-Datenbanksystem, natives
XML-Datenbanksystem, das vorrangig die Speicherung und Verarbeitung von *XML-Daten* über XML-orientierte Schnittstellen unterstützt (im Gegensatz zu den um XML-Unterstützung erweiterten relationalen, objektrelationalen oder objektorientierten *Datenbanksystemen*).

XML-Dokument
XML-Datenobjekt, das gemäß der *W3C Recommendation* XML 1.0 *wohlgeformt* ist.

XML-Dokument, datenorientiertes
XML-Dokument, dessen Inhalt meist feingranular und sehr regulär strukturiert ist und auf einem *Schema* mit Typinformationen basiert. Die Ordnung von Geschwisterelementen ist in der Regel nicht signifikant, Elemente haben den Typ *element content*.

XML-Dokument, dokumentorientiertes
XML-Dokument, das grobgranular mit irregulärer Struktur vorliegt, häufig kein explizites Schema besitzt und Elemente vom Typ *mixed content* enthält. Die Reihenfolge der Geschwisterelemente ist signifikant.

XML-Dokument, gemischt strukturiertes
Mischform aus *dokumentorientiertem* und *datenorientiertem XML-Dokument*.

XML-Instanz
Gültiges oder *schemagültiges XML-Dokument*.

XML-Objekt siehe *XML-Dokument*

XML-Prozessor
Software-Modul zur Durchmusterung (*Parser*), Validierung (*wohlgeformt*) und Verifikation (*gültig* bzw. *schemagültig*) von *XML-Dokumenten*. XML-Prozessoren sind u.a. *DOM*-Implementationen und das *SAX*.

XML-QL
Früher Vorschlag für eine XML-Anfragesprache (siehe Kap. 3).

XML Schema
W3C Recommendation zur Spezifikation von *Schemata* für *XML-Dokumente* (siehe Kap. 2).

XML-Speicherung siehe *Speicherung von XML-Dokumenten*

XPath
Sprache zur Navigation in *XML-Dokumenten*. Version 1.0 (*W3C Recommendation*) vom November 1999, aktueller Working Draft in der Version 2.0 (siehe Kap. 3).

XQL
Früher Vorschlag für eine XML-Anfragesprache (siehe Kap. 3).

XQuery
Sprache zur Formulierung von Anfragen an *XML-Dokumente*, verbindet *XPath* als Navigationssprache mit dem *SQL*-artigen FOR-LET-WHERE-RETURN (FLWR)-Klauselkonstrukt. Die Standardisierung erfolgt im Rahmen des *W3C* (siehe Kap. 3).

XSLT (eXtensible Stylesheet Language Transformations)
Funktionale, regelbasierte Sprache zur Transformation von *XML-Dokumenten*. Spezifikation liegt als *W3C Recommendation* vor (siehe Kap. 3).

XUpdate
Vorschlag der XML:DB-Initiative für einen Standard zur Manipulation von *XML-Daten*. Siehe auch http://www.xmldb.org/xupdate.

Die Autoren

Wolfgang Benn ist Professor für Datenverwaltungssysteme an der TU Chemnitz. Nach dem Studium der Informatik in Hamburg und anschließender Promotion folgte eine Anstellung in der Industrie im Bereich wissensbasierter Systeme. Weitere Stationen waren die FernUniversität in Hagen, die Universität Duisburg und kurzzeitig auch die Universität Dortmund, bevor 1992 der Ruf an die Technische Universität Chemnitz erfolgte. Arbeitsschwerpunkte sind intelligente Datenbanken, neuronal basierte, mehrdimensionale Indexierungstechniken für Datenbanken, Datenbanken für subsymbolische Massendaten und die Entwicklung semistrukturierter Middleware. Die Professur Datenverwaltungssysteme ist seit dem Jahr 2000 am Sonderforschungsbereich 457, Regionale hierarchielose Produktionsnetze, der Deutschen Forschungsgemeinschaft beteiligt.

Timo Böhme legte sein Diplom in Informatik 1999 an der Universität Leipzig mit Auszeichnung ab und gehört seitdem als wissenschaftlicher Mitarbeiter der Arbeitsgruppe Datenbanken im Institut für Informatik der Universität Leipzig an. Seine derzeitigen Forschungsinteressen liegen im Bereich XML und Integration von XML in Datenbanksysteme.

Ilvio Bruder ist Promotionsstipendiat an der Universität Rostock am Lehrstuhl für Datenbank- und Informationssysteme. Sein Diplom legte er 2000 ab. Danach arbeitete er an Forschungsprojekten als wissenschaftlicher Mitarbeiter am Lehrstuhl Datenbank- und Informationssysteme. Seine Forschungsinteressen liegen unter anderem im Bereich Multimedia Information Retrieval, Web-Suchmaschinen und wissensbasierten Systemen.

Stefan Conrad ist Professor für Praktische Informatik an der Ludwig-Maximilians-Universität München, Lehr- und Forschungseinheit Datenbanksysteme. Zurzeit ist er ferner Sprecher des Arbeitskreises »Grundlagen von Informationssystemen« im GI-Fachbereich DBIS. Nach dem Informatikstudium und der Promotion (1994) an der TU Braunschweig war er Assistent an der Universität Magdeburg. Dort habilitierte er 1997 mit einer Schrift über Föderierte Datenbanksysteme (als Buch im Springer-Verlag erschienen). Seine Forschungsschwerpunkte sind die Integration heterogener Datenbanken, Föderierte Systeme und der Entwurf von Informationssystemen.

Sven Helmer ist wissenschaftlicher Assistent am Lehrstuhl für Praktische Informatik III an der Universität Mannheim, an der er auch im Jahr 2000 promovierte. Seine Forschungsinteressen liegen im Bereich Indexstrukturen für postrelationale Datenbanksysteme, Anfrageoptimierung und Implementierung von Data Warehouses und nativen XML-Repositories.

Andreas Heuer promovierte 1988 und habilitierte 1993 an der TU Clausthal. Er nahm im Jahre 1994 einen Ruf an die Universität Rostock auf den Lehrstuhl Datenbank- und Informationssysteme an. Er lehrt und forscht seitdem in den Gebieten Datenbanken, objektorientierte Systeme und digitale Bibliotheken. Er ist Sprecher der Fachgruppe Datenbanksysteme und des Arbeitskreises Digitale Bibliotheken der Gesellschaft für Informatik (GI) sowie stellvertretender Sprecher des Fachbereiches Datenbank- und Informationssysteme der GI. In über zehn Forschungsprojekten hat er objektrelationale und objektorientierte Datenbanksysteme eingesetzt oder selbst weiterentwickelt. Zu den aktuellen Projekten gehören föderierte Informationssysteme, Analyse- und Suchdienste für Texte im Internet (basierend auf Suchmaschinen, objektrelationalen Datenbanken und XML-Datenbanken) sowie Dokumentdatenbanken für verteilte digitale Bibliotheken.

Peter Jaeschke forschte nach dem Studium des Wirtschaftsingenieurwesens von 1991 bis 1994 als wissenschaftlicher Mitarbeiter am Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB) der Universität Karlsruhe (TH); 1995 promovierte er bei Wolffried Stucky. Im Rahmen einer Kooperation zwischen Promatis und dem AIFB agierte er bereits während der Zeit von 1991 bis 1994 im Auftrag von Promatis als leitender Berater in verschiedenen Projekten. Von 1994 bis 1996 war er als Projektmanager für Promatis tätig. 1996 wurde er zum geschäftsführenden Gesellschafter der Promatis Consulting AG, Zürich berufen und wurde damit Mitglied des Promatis Executive Boards. Seit Februar 2001 hat Dr. Jaeschke als Executive Vice President Software Products die Gesamtverantwortung für den Kernprozess Software-Produkte, von der Produktentwicklung über das Produktmanagement bis zum Produktmarketing und -vertrieb. Als Verwaltungsratspräsident ist er verantwortlich für den gesamten Geschäftsbetrieb der Promatis Consulting AG, Zürich. Seit 1999 ist er Mitglied des Promatis Management Teams.

Gerti Kappel war Professorin für Informationssysteme an der Johannes-Kepler-Universität Linz und ist seit Oktober 2001 Professorin für Wirtschaftsinformatik an der Technischen Universität Wien und Leiterin der Business Informatics Group am Institut für Softwaretechnik und Interaktive Systeme. Sie beschäftigt sich in Forschung und Lehre mit objektorientierter Softwareentwicklung, Datenbank/Web-Integration und ubiquitärer Web-Technologie, sowie deren Anwendung in Workflow Management und Electronic Commerce. Sie ist u.a. Koautorin des Buchs »UML@Work – Von der Analyse zur Realisierung« (dpunkt.verlag) und wissenschaftliche Beraterin in vielen Projekten gemeinsam mit der Industrie.

Markus Keidl studierte von 1994 bis 1999 Informatik an der Universität Passau und arbeitet dort seit 1999 als wissenschaftlicher Mitarbeiter am Lehrstuhl für Dialogorientierte Systeme. Seine Arbeitsschwerpunkte liegen in den Bereichen verteilte Datenbanksysteme, Sicherheit in verteilten Systemen, Metadaten und Publish/Subscribe-Systeme sowie Web Services.

Alfons Kemper ist seit 1993 Inhaber des Lehrstuhls für Dialogorientierte Systeme an der Universität Passau. Davor war er von 1991 bis 1993 Professor an der RWTH Aachen und von 1984 bis 1991 Hochschulassistent an der Universität Karlsruhe, wo er sich 1991 auch habilitierte. Sein Studium der Informatik begann er 1977 an der Universität Dortmund, wechselte aber nach dem Vordiplom an die University of Southern California in Los Angeles, wo er den Master of Science und den Ph. D.-Abschluss in den Jahren 1981 bzw. 1984 erwarb. Er hat neben zahlreichen Forschungsartikeln auch zwei Lehrbücher über Datenbanksysteme geschrieben.

Meike Klettke studierte Informatik an der Universität Rostock und promovierte dort 1997 im Bereich Datenbanken. Seit 1992 ist sie wissenschaftliche Mitarbeiterin an der Universität Rostock/TU Cottbus und arbeitet auf den Gebieten Datenbank-Entwurf, Reverse-Engineering von Datenbanken sowie Datenbankentwicklung für Suchmaschinen. Aktueller Forschungsgegenstand sind Technologien im Bereich XML und Datenbanken.

Oliver Langer hat in Chemnitz Informatik studiert und war bis Mai 2002 wissenschaftlicher Mitarbeiter an der Professur Datenverwaltungssysteme der TU Chemnitz. Während des Studiums war er in den Bereichen parallele Rechnerarchitekturen und Künstliche Intelligenz tätig. Danach lagen seine Arbeitsschwerpunkte in der Konzeption und Entwicklung eines semistrukturierten Modellkerns für den Einsatz als industrielle Middleware. Seit Juni 2002 ist er als Consultant bei CoreMedia in Hamburg tätig.

Georg Lausen legte sein Diplom 1978 in Wirtschaftsingenieurwesen (Schwerpunkt Informatik und Operations Research) an der Universität Karlsruhe (TH) ab. Daran anschließend war er bis 1986 wissenschaftlicher Angestellter und später Hochschulassistent am Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB) der Fakultät für Wirtschaftswissenschaften der Universität Karlsruhe (TH). Er promovierte 1982 zum Dr. rer. pol. und habilitierte sich 1985 für das Fach Angewandte Informatik. Von 1986 bis 1987 war er Professor für das Gebiet *Informationstechnologie und ihre Integrationsproblematik* am Fachbereich Informatik der TH Darmstadt und daran anschließend bis 1994 Professor für *Praktische Informatik* an der Fakultät für Mathematik und Informatik der Universität Mannheim. Derzeit hat er eine Professur für *Informatik* an der Fakultät für Angewandte Wissenschaften der Universität Freiburg inne. Er erhielt 1999 den ACM SIGMOD Test of Time Award (gemeinsam mit M. Kifer) für

die Arbeit *F-logic: A higher-order language for Reasoning about Objects, Inheritance and Scheme*. Sein derzeitiger Interessenschwerpunkt ist die Entwicklung und der praktische Einsatz von XML-basierten Technologien.

Wolfgang May legte sein Diplom in Informatik 1995 an der Universität Karlsruhe (TH) mit Auszeichnung ab. Seit Mai 1995 gehört er der Arbeitsgruppe *Datenbanken und Informationssysteme* am Institut für Informatik der Universität Freiburg an, zunächst bis 1997 als Stipendiat im Graduiertenkolleg *Menschliche und Maschinelle Intelligenz*, dann als wissenschaftlicher Mitarbeiter, und seit seiner Promotion 1998 als wissenschaftlicher Assistent. Mit der Habilitationsschrift »*A Logic-Based Approach to XML Data Integration*« erhielt er im Dezember 2001 die Lehrbefugnis für Informatik. Seine derzeitigen Forschungsinteressen liegen unter anderem im Bereich Logik und XML.

Holger Meyer studierte Automatisierungstechnik und Technische Kybernetik an der Universität Rostock. 1990 promovierte er auf dem Gebiet der Computertechnik mit einer Arbeit zu Echtzeit-Datenbanksystemen. Seit 1992 ist er wissenschaftlicher Oberrat am Fachbereich Informatik der Universität Rostock. Dr. Meyer lehrt und forscht in den Gebieten Datenbanken, Anfrageoptimierung, Verteilte Datenbanken und WWW. Er hat ein DFG-Projekt zur Anfrageoptimierung und Lastverteilung in Verteilten Datenbanken geleitet und an zahlreichen Grundlagen- und Anwendungsprojekten mitgearbeitet. Sein aktueller Forschungsgegenstand ist die Verwaltung und Verarbeitung von semistrukturierten Dokumenten, speziell XML-Dokumenten, für digitale Bibliotheken.

Guido Moerkotte leitet seit 1996 die Datenbankgruppe an der Universität Mannheim. Seine Interessensgebiete umfassen Data Warehouses, Anfrageoptimierung und Indexstrukturen. Seit Anfang 1998 befasst er sich mit Datenbanksystemen, die speziell für XML entwickelt werden. Im Rahmen dieser Arbeiten entstand das inzwischen kommerziell erhältliche XML-Datenbanksystem Natix.

Andreas Oberweis erhielt sein Diplom in Wirtschaftsingenieurwesen von der Universität Karlsruhe im Jahr 1984. Von 1985 bis 1995 war er als Assistent an den Universitäten Darmstadt, Mannheim und Karlsruhe tätig. Er promovierte im Jahr 1990 am Fachbereich Mathematik und Informatik der TH Darmstadt, und habilitierte sich 1995 an der Fakultät für Wirtschaftswissenschaften der Universität Karlsruhe. Seit 1995 ist er Inhaber des Lehrstuhls für Entwicklung betrieblicher Informationssysteme an der Universität Frankfurt. Seine Hauptforschungsinteressen sind Geschäftsprozessmanagement, Informationssystem-Engineering und Software Engineering Management. In der GI ist Prof. Oberweis Sprecher der Fachgruppe »Entwicklungsmethoden für Informationssysteme und deren Anwendung«; ferner ist er Mitgründer der Promatis AG (1990) und der Virtual Global University VGU (2001).

Birgit Pröll promovierte 2001 nach dem Studium der Informatik an der Johannes-Kepler-Universität Linz im Bereich Web-basierter Informationssysteme. Seit 1991 ist sie wissenschaftliche Mitarbeiterin am Institut Für Anwendungsorientierte Wissensverarbeitung (FAW). Von 1996 bis 2001 hatte sie die Leitung des FAW-Projektteams inne, das die Entwicklung des Österreich-weiten Web-basierten Tourismusinformationssystems TIScover durchführt. Ihre Forschungsschwerpunkte liegen in den Bereichen Information Retrieval, Web Engineering, Entwicklung ubiquitärer Web-Anwendungen und Electronic Commerce.

Erhard Rahm ist seit 1994 Professor für Datenbanken an der Universität Leipzig. Er promovierte und habilitierte am Fachbereich Informatik der Universität Kaiserslautern und verbrachte längere USA-Forschungsaufenthalte bei IBM Research und Microsoft Research. Seine Arbeitsgebiete umfassen Data Warehousing (u.a. in der Bioinformatik), Metadaten-Verwaltung, Workflow-Management und Performance-Bewertung und -Optimierung von Informationssystemen. Er ist Autor bzw. Koautor zahlreicher Publikationen und mehrerer Bücher, u.a. zu Mehrrechner-Datenbanksystemen und zur Datenbanksystem-Implementierung. Prof. Rahm ist Mitglied des Leitungsgremiums des GI-Fachbereichs Datenbanken und Informationssysteme, Sprecher des 2001 gegründeten Arbeitskreises »Web und Datenbanken« und Tagungsleiter der BTW 2003 (10. GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web) in Leipzig.

Werner Retschitzegger promovierte 1996 nach dem Studium der Wirtschaftsinformatik an der Johannes-Kepler-Universität Linz im Bereich aktiver objektorientierter Datenbanktechnologien. Seit 1993 ist er wissenschaftlicher Mitarbeiter an der Abteilung für Informationssysteme der Johannes-Kepler-Universität Linz. Er habilitierte sich 2000 für das Fach Angewandte Informatik im Bereich Datenbanktechnologien für Nicht-Standardanwendungen. Derzeit hat er eine Vertretungsprofessur für Wirtschaftsinformatik an der Technischen Universität Wien inne. Seine Forschungsschwerpunkte liegen in den Bereichen Datenbanken und Web, Interorganisationale Workflows, Ubiquitäre Web-Anwendungen sowie Electronic Commerce.

Gunter Saake ist Professor für Datenbanken und Informationssysteme an der Otto-von-Guericke-Universität Magdeburg und forscht unter anderem auf den Gebieten Datenbankintegration, digitale Bibliotheken, objektorientierte Informationssysteme und Informationsfusion. Er ist Koautor mehrerer Lehrbücher, u.a. zu Datenbanken und Java, Datenbankkonzepten und -implementierungstechniken, sowie Herausgeber der Zeitschrift »Datenbank-Spektrum«.

Kai-Uwe Sattler ist wissenschaftlicher Assistent in der Arbeitsgruppe Datenbanken der Otto-von-Guericke-Universität Magdeburg und vertritt gegenwärtig den Lehrstuhl Datenbanken an der TU Dresden. Seine Arbeitsgebiete sind Datenbankintegration und -föderation, Internet-Datenbanken sowie speziell Anfrage-

bearbeitung in heterogenen Datenbanken. Er ist Koautor mehrerer Lehrbücher zu Datenbanken sowie Algorithmen und Datenstrukturen. Weiterhin ist er Herausgeber der Zeitschrift »Datenbank-Spektrum«.

Harald Schöning studierte Informatik an der Universität Kaiserslautern und promovierte dort 1992 am Lehrstuhl von Prof. Härder. Seit 1993 arbeitet er in der Zentrale der Software AG in Darmstadt, zunächst als Entwickler für das Datenbanksystem Adabas, später als Projektleiter. Momentan ist er Architekt für das XML-Datenbanksystem Tamino. Für die Deutsche Informatik-Akademie gibt Dr. Schöning regelmäßig Seminare zu XML-bezogenen Themen.

Stefan Seltzsam studierte von 1994 bis 1999 Informatik an der Universität Passau. Er erwarb sein Diplom 1999, für welches er mit dem Fakultätspreis der Fakultät für Mathematik und Informatik ausgezeichnet wurde. Seitdem ist er wissenschaftlicher Mitarbeiter am Lehrstuhl für Dialogorientierte Systeme. Seine Forschungsinteressen liegen in den Bereichen verteilte Datenbanksysteme, Sicherheit in verteilten Systemen und Web Services.

Konrad Stocker studierte von 1993 bis 1997 Informatik und Betriebswirtschaftslehre an der Universität Passau. Er erwarb sein Diplom 1997, für welches er mit dem Fakultätspreis der Fakultät für Mathematik und Informatik ausgezeichnet wurde. Im Anschluss an seine Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Dialogorientierte Systeme promovierte er 2001 auf dem Gebiet der Anfrageoptimierung in verteilten Datenbanken. Derzeit ist er bei der BMW Group in München beschäftigt.

Thomas Stöhr studierte Informatik an der Universität Kaiserslautern (Diplom 1993) und arbeitete als wissenschaftlicher Mitarbeiter an den Datenbank-Lehrstühlen der Universität Kaiserslautern (Prof. Härder, 1993-1994) und der Universität Leipzig (Prof. Rahm, 1994-1999). Er ist aktuell Doktorand am Leipziger Lehrstuhl und beschäftigt sich im Rahmen seiner Promotion mit Performance-Aspekten auf den Gebieten Parallele Datenbanksysteme und Data Warehousing, wobei er nationale und internationale Publikationen erzielen konnte. Im Rahmen von Industrieprojekten war er für die Entwicklung und praktische Umsetzung von Evaluierungsverfahren für unterschiedlichste kommerzielle Werkzeuge (Data Warehouse, Business Intelligence) verantwortlich.

Rainer Unland ist Professor für Datenverwaltungssysteme und Wissensrepräsentation am Institut für Informatik an der Universität Essen. Die Interessen des Lehrstuhls liegen in der Schnittstelle von XML, Datenbanksystemen und dem Web, der Agententechnologie und der framework- und komponentenbasierten Softwareentwicklung.

Bahram Vojdani ist wissenschaftlicher Mitarbeiter am Lehrstuhl für Datenverwaltungssysteme und Wissensrepräsentation am Institut für Informatik an der Universität Essen. Seine Forschungsschwerpunkte liegen in der Schnittstelle zwischen XML, Datenbanksystemen und dem Web.

Gottfried Vossen ist Professor für Informatik, insbesondere Datenbanken und Informationssysteme am Institut für Wirtschaftsinformatik der Universität Münster. Er studierte, promovierte und habilitierte sich an der RWTH Aachen und hatte Gastprofessuren inne an der University of California in San Diego, USA, an der Karlstad Universitet in Schweden sowie an den Universitäten in Kiel, Düsseldorf und Koblenz. 1991 wurde er an die Universität Gießen, 1993 nach Münster berufen; im Rahmen eines Forschungssemesters war er im Jahr 2002 bei der Promatis Corp. in Kalifornien tätig. Seine Forschungsinteressen sind konzeptionelle sowie anwendungsnahe Fragestellungen im Umfeld von Datenbanken, Informationssystemen und Workflow-Management; Schwerpunkte liegen derzeit in den Bereichen Data-Warehouse-Entwurf, Datenbankmetasprachen, Prozess-Modellierung, E-Learning und XML. Dr. Vossen ist Mitglied des Herausgeberrats der Zeitschriften *Information Systems* und *International Journal of Computational and Numerical Analysis and Applications*.

Walter Waterfeld studierte an der TU Darmstadt Informatik und promovierte anschließend am dortigen Datenbank-Lehrstuhl. Seit 1990 arbeitet er in verschiedenen Positionen in der Produktentwicklung der Software AG. Unter anderem war er Architekt und Projektleiter für die objektorientierten Erweiterungen der Datenbanktechnologie und für die Persistenzkomponente von Bolero, einer Entwicklungs- und Ablaufumgebung für kommerzielle Web-Anwendungen der Software AG. Seit 2000 ist er Architekt für das XML-Datenbanksystem Tamino.

Gunnar Weber ist seit April 2000 wissenschaftlicher Assistent am Lehrstuhl Datenbank- und Informationssysteme an der Universität Rostock. Nach Beendigung seines Informatikstudiums 1998 war er zunächst als wissenschaftlicher Mitarbeiter an Projekten in den Bereichen Internet-Suchdienste und Föderierte Informationssysteme beteiligt. Seine Forschungsinteressen sind die Integration von Informationssystemen in Suchmaschinen sowie Anfragesprachen, Indexstrukturen und Anfrageverarbeitung für Suchdienste auf semistrukturierten Daten.

Gerhard Weikum ist seit 1994 Professor für Informatik an der Universität des Saarlandes in Saarbrücken. Er hat 1986 in Darmstadt promoviert und war seitdem bei MCC in Austin, an der ETH Zürich sowie – im Forschungssemester – bei Microsoft Research in Redmond tätig. Seine Arbeitsgebiete umfassen Leistungs- und Optimierungsaspekte verteilter Informationssysteme, Workflow-Management und E-Services sowie intelligentes Suchen auf semistrukturierten Daten. Er ist Mitglied des Herausgebergremiums der Zeitschrift *ACM Transactions on*

Database Systems, Mitglied des Leitungsgremiums des GI-Fachbereichs Datenbanken und Informationssysteme, Vizepräsident der VLDB Endowment und war bzw. ist in zahlreichen weiteren wissenschaftlichen Gremien aktiv.