

Datenbanken und XML

Oliver Klein

13. Juni 2001

Zusammenfassung

Die zunehmende Vernetzung von Rechnersystemen und die steigende Nutzung des WWW für den Informationsaustausch machen einfache und durch Applikationen leicht zu verarbeitende Austauschformate erforderlich. XML wird immer häufiger zum Standardformat für die Repräsentation sowohl strukturierter als auch unstrukturierter Daten.

XML ermöglicht den Datenaustausch zwischen Anwendungen, WWW und Datenbanken und wird bereits in einer Vielzahl von unterschiedlichen Produkten eingesetzt. Dieser Beitrag gibt einen Überblick über die Fähigkeiten und Kategorien, zu denen diese Produkte zuzuordnen sind. Außerdem werden mehrere Modelle vorgestellt, um XML-Strukturen auf Datenbanken abzubilden, um einerseits bestehende Datenbankinhalte nach XML zu konvertieren und andererseits die Fähigkeiten von DBMS zu nutzen, um XML-Daten speichern und Anfragen einfach realisieren zu können.

Inhaltsverzeichnis

1	Einleitung	3
1.1	Motivation	3
1.2	Datenbanken und XML	3
1.3	Aufbau	3
2	XML	4
2.1	XML-Terminologie	4
2.2	Dokumenttypdefinitionen	5
2.3	XML Erweiterungen	5
2.4	Einsatzbereiche von XML	6
2.5	Speicherung von XML-Dokumenten und -Daten	6
3	XML-Daten in Datenbanken	7
3.1	Produktkategorien	8
3.1.1	Middleware	8
3.1.2	XML-fähige Datenbanken	9
3.1.3	Native XML-Datenbanken	9
3.1.4	XML (Application-) Server	10
3.1.5	Content Management Systeme	10
3.1.6	Persistente DOM Implementierungen	11
3.2	Abbildungen zwischen XML und Datenbanken	11
3.2.1	Template-Driven Mapping	12
3.2.2	Model-Driven Mappings	12
3.2.3	DTDs und Datenbankschemata	16
3.2.4	Alternative Abbildungen	17
4	Beispiele für XML verarbeitende Datenbankmanagementsysteme	18
4.1	XML-fähige Datenbanken am Beispiel Oracle8i	18
4.2	XML Server am Beispiel Tamino	19
5	Abschließende Anmerkungen	20

1 Einleitung

1.1 Motivation

Die Vernetzung von Rechnersystemen sowie die steigende Nutzung des WWW für den Informationsaustausch machen Austauschformate erforderlich, die einerseits verständlich, andererseits aber auch leicht durch Applikationen verarbeitet werden können. Ein solches Format soll XML (eXtensible Markup Language) bieten [BPSM00]. XML-Dokumente sind selbstbeschreibend, einfach strukturierbar und bieten sich somit für die Repräsentation von Daten aus den unterschiedlichsten Anwendungsgebieten an. Bei einem zunehmendem Einsatz von XML lässt sich aber schnell ein einfacher Anspruch ableiten: XML-Dokumente müssen gezielt nach Informationen durchsucht werden können, um auch bei großen Datenmengen handhabbar zu sein. Der Umgang mit großen Mengen strukturierter Daten ist traditionell das Gebiet der Datenbankmanagementsysteme. Dieser Artikel geht unter anderem der Frage nach, wie die klassischen Datenbanktechnologien und XML miteinander verbunden werden können.

1.2 Datenbanken und XML

XML wird häufig als Austauschformat für Daten angewendet. Für die Speicherung der Informationen und für die Realisierung von Anfragen bieten sich unterschiedliche Möglichkeiten an. XML-Dokumente können im Dateisystem eines Rechners, aber auch in relationalen Datenbanken, objektorientierten Datenbanken oder spezialisierten Systemen gespeichert werden. Dateien bieten dabei allerdings die ungeeignetste Alternative, da lediglich statische Dokumente gespeichert werden können und Anfragen nur ineffizient realisierbar sind. XML-Strukturen können dagegen recht einfach auf objektorientierte Datenbanken abgebildet werden. Bei der Speicherung und Rekonstruktion von XML-Dokumenten in relationalen Datenbanken existieren unterschiedliche Ansätze, von denen einige hier vorgestellt werden. Als Vorteilhaft erweisen sich dabei auch die Möglichkeiten, über SQL-Anfragen sehr schnell XML-Dokumente zu durchsuchen sowie Alt-Daten in XML zu konvertieren bzw. zu veröffentlichen.

Auf die Verwaltung von semistrukturierten Daten bzw. XML-Dokumenten spezialisierte Systeme wie z.B. LORE sind im kommerziellen Bereich weniger verbreitet. Eine neue Gruppe von sogenannten nativen XML Datenbanken, die speziell auf die Bedürfnisse von XML zugeschnitten sind, findet jedoch zunehmend Verbreitung. Die unterschiedlichen Ansätze und die damit verbundenen Probleme werden in Kapitel 3 genauer betrachtet.

1.3 Aufbau

Nach einem kurzen Überblick über XML, weiterführende Techniken wie DTDs etc. sowie Speichermöglichkeiten wird in diesem Artikel zunächst eine kategorische Einteilung von XML Datenbank Produkten vorgenommen (vergl. [Bou00b]). Es werden Modelle vorgestellt, um Daten zwischen XML und Datenbanken bzw. ihren Schemata abbilden zu können, wobei die Einsatz von DTDs ebenfalls eine Rolle spielt. Anhand der Beispiele Oracle8i und Tamino (Software AG) wird im letzten Abschnitt gezeigt, wie kommerzielle relationale Datenbanken und spezialisierte Systeme die zuvor erläuterten Modelle unterstützen und welche Unterstützung diese beiden Systeme beim Einsatz von XML bieten.

2 XML

XML steht für eXtensible Markup Language. Als Standard des World Wide Web Consortiums (W3C) dient XML im Gegensatz zu HTML zur Beschreibung und Identifikation von Daten und Inhalten im World Wide Web. HTML eignet sich in erster Linie zur Präsentation und Darstellung von Webseiten innerhalb eines Browsers und verwendet spezielle vordefinierte Markups bzw. Tags. Ebenso wie HTML basiert auch XML auf der bereits etwas älteren SGML (Structured Generalized Markup Language), wobei HTML aber eine konkrete Anwendung und XML eine Untermenge von SGML ist [BPSM00]. Im Unterschied zu HTML sind bei XML die Tags demnach nicht statisch. Sie sind vom Autoren eines XML-Dokuments frei wählbar und sollen die Daten zwischen einem öffnenden (Start-) und einem schließenden (Ende-) Tag semantisch beschreiben. Der Vorteil einer solchen semantischen Definition liegt darin, dass die Daten sowohl vom Menschen als auch von Maschinen bzw. Programmen interpretiert und ausgewertet werden können, sofern ein sinnvolles Verständnis der Bedeutung der gewählten Tags gegeben ist.

2.1 XML-Terminologie

XML-Dokumente bestehen aus einem oder mehreren Elementen. Ein Element, wie zum Beispiel `<name>Bradley</name>`, beginnt immer mit einem öffnenden Tag und endet mit einem schließenden Tag. Der Text zwischen den Tags wird als Teil des Elements angesehen und stellt den Inhalt bzw. die eigentliche Information dar. Die Tagnamen sagen etwas über die Bedeutung des Inhalts aus. Ähnlich wie in HTML können auch in XML Attribute definiert werden, die jeweils im öffnenden Tag eines Elements zusätzliche Informationen bereitstellen. Bei leeren Elementen, die nur nichttextuelle Information bereitstellen, können beide Tags zusammengefasst werden:

```
<Picture src="blueball.gif"></Picture>
```

```
<Picture src="blueball.gif"/>
```

Die Elemente innerhalb eines Dokuments können beliebig tief verschachtelt werden, um die Struktur der Daten widerzuspiegeln. Dabei ist darauf zu achten, dass das zuletzt geöffnete Element auch als erstes wieder geschlossen wird, damit ein Dokument im Sinne der XML-Spezifikation gültig ist.

Aufgrund der semantischen Auszeichnung der Daten, sowie der Möglichkeiten zur Strukturierung der Informationen durch Attribute, Elemente und Verschachtelung von Elementen in einem Dokument, eignet sich XML besonders zur Beschreibung von semistrukturierten Daten¹. Semistrukturierte sowie strukturierte Daten lassen sich in XML wie im folgenden Beispiel darstellen:

¹Semistrukturierte Daten besitzen kein vorgegebenes Datenschema. Ein einzelnes Datum wird durch ein *Label-Value*-Paar repräsentiert, wobei das Label eine Beschreibung (Metainformation) für den Value, also den Wert (Nutzdatum) ist. Die Nutzdaten können wiederum aus Label-Value-Paaren bestehen. Eine mögliche Notation kann z.B. wie folgt aussehen: {person: {name: 'Alan', phone: '3127786', email: 'agg@abc.com'}}. (vergl. [ABS00])

```

<buch isbn='0-201-342855' >
  <titel>
    The XML companion
  </titel>
  <autor>
    <name>Bradley</name>
    <vorname>Neil</vorname>
  </autor>
</buch>

```

Für die semantische Interpretation der Daten ist es notwendig, dass ein einheitliches Verständnis der XML-Tags und deren Verwendung gegeben ist. Die syntaktische Definition der Tags erfolgt dabei entweder mit ihrem erstmaligen Auftreten im Dokument oder durch die formale Deklaration der verwendeten XML-Elemente in einer Document Type Definition (DTD).

2.2 Dokumenttypdefinitionen

In einer DTD wird spezifiziert, wie XML-Elemente, Attribute und andere Daten definiert sind, und mit welchem logischen Bezug zueinander sie innerhalb des Dokuments verwendet werden können. Eine DTD stellt somit für das Dokument und seine Elemente eine kontextfreie Grammatik bereit, die jederzeit herangezogen werden kann, um XML-Dokumente auf ihre Gültigkeit zu überprüfen. Das folgende Beispiel zeigt eine DTD zum vorangegangenen XML-Ausschnitt:

```

<!ELEMENT buch (titel, autor+) >
<!ATTLIST isbn #REQUIRED>
<!ELEMENT titel (#PCDATA) >
<!ELEMENT autor (name, vorname) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT vorname (#PCDATA) >

```

XML-Dokumente können ohne eine DTD nur auf *Wohlgeformtheit* gemäß der XML-Spezifikation überprüft werden, sie werden bei Korrektheit als *well-formed* bezeichnet. Genügt ein Dokument dagegen auch seiner DTD, ist es *valid*.

Häufig kommt es vor, dass DTDs mit Datenschemata für XML-Dokumente verwechselt werden. Der klassische Schemabegriff, wie er auch auf dem Gebiet der Informationssysteme verwendet wird, unterscheidet sich hier aber in wesentlichen Punkten. XML bzw. die DTD kennt keine differenzierten atomaren Datentypen außer PCDATA, welcher als Zeichenkette mit eingeschränkter Syntax interpretiert wird (um nicht mit Tags verwechselt zu werden) und ANY für beliebige Zeichen. Außerdem spielt bei der DTD innerhalb der Verschachtelung die Reihenfolge der Subelemente eine wichtige Rolle.

2.3 XML Erweiterungen

Das W3C arbeitet auch an einer eigenen Schema-Sprache für XML. XML-Schema ist eine Erweiterung des DTD-Konzepts. Sie erlaubt eine differenzierte Typisierung der Elemente sowie Einschränkungen für Wertebereiche [CT00].

Für eine von Struktur und Inhalt unabhängige Darstellung von XML-Dokumenten stehen unterschiedliche Technologien zur Verfügung. Beispielsweise können für die Präsentation in einem Browser Cascading Style Sheets (CSS) [Bos00] mit dem Dokument verknüpft, oder auch die eXtensible Stylesheet Language XSL z.B. für die Transformation von XML nach HTML verwendet werden [Fro00].

Die vielfältigen Konzepte und Eigenschaften machen XML zu einem in den unterschiedlichsten Bereichen einsetzbaren Format, das selbstbeschreibend, flexibel, erweiterbar, plattform- und herstellerunabhängig ist. An der Weiterentwicklung des Standards sind verschiedene Gruppen beteiligt, was letztlich der Verbreitung von XML in vielfältigen Anwendungen und Systemen zugute kommt.

2.4 Einsatzbereiche von XML

Die immer stärker werdende Vernetzung von Rechnern und der damit verbundene Anstieg des elektronischen Informationsaustausches erfordert leicht anwendbare Austauschformate. XML wird aufgrund der oben beschriebenen Eigenschaften immer häufiger in unterschiedlichen Bereichen eingesetzt:

- EDI (Electronic Data Interchange)
- Verbindung verschiedener Softwarekomponenten
- Anbindung von Datenbanken an Applikationen
- Import/ Export von Datenbankinhalten

XML-Dokumente können semistrukturierte bis hoch strukturierte Daten enthalten. Aufgrund ihres selbstbeschreibenden Charakters sind Aufgaben wie Filtern, Restrukturierung, Analyse, Anfragen, automatische Generierung, Rekonstruktion und die gezielte Informationssuche leichter auf XML-Dokumente anwendbar.

2.5 Speicherung von XML-Dokumenten und -Daten

Wie bei HTML, so wird auch der größte Teil der XML-Dokumente mit statischen Inhalten in Form von Dateien innerhalb eines Dateisystems gespeichert. Für die Speicherung und Verwaltung von großen Informationsmengen in XML, die gezielt durchsucht werden sollen, empfehlen sich (Datenbank-)Managementsysteme mit unterschiedlichen Ansätzen bei Speicherung und Anfragen an XML-Daten.

Für XML-Dokumente mit irregulärer Struktur, d.h. mit uneinheitlichen textuellen Abschnitten, welche die Informationen enthalten, nur wenigen Tags im Verhältnis zum Gesamtdokument und wenigen gleichartigen Elementen, bieten sich sogenannte Content-Management-Systeme an [Bou00a]. Sie unterstützen den Anwender beim Verfassen und Verwalten von inhaltsbezogenen XML-Dokumenten, bieten Funktionen wie Mehrbenutzerfähigkeit, Versionskontrolle, integrierte Editoren und Volltextrecherche. Ein Beispiel für ein solches System ist die POET

Content Management Suite (CMS) [Sör00]. Das System ist als Server für komplexe Medientypen konzipiert worden und kann neben XML und SGML auch mit Grafik-, Audio- und Videoformaten umgehen. POET speichert XML-Dokumente intern in einer Objektorientierten Datenbank. Wegen der hierarchischen Struktur von XML eignen sich OO-Datenbanken besonders zur Speicherung und Verwaltung von XML-Daten. Der eXcelon Explorer, eine OO-Datenbank für XML, stellt XML-Dokumente intern als XML-Datentypen dar und unterstützt als XML-Anfragesprache XQL. Relationale und objektrelationale DBMS, wie z.B. DB2 von IBM oder Oracle8i, bilden XML-Strukturen auf Relationen und Objekte ab und ermöglichen es, in herkömmlicher Weise Anfragen in SQL zu formulieren. Das System LORE (Lightweight Object Repository) der Stanford University wurde ursprünglich für den Umgang mit semistrukturierten Daten entwickelt und verfügt über eine eigene komfortable und mächtige Anfragesprache für XML namens Lorel.

3 XML-Daten in Datenbanken

Große Mengen von Daten und Informationen werden idealer Weise in Datenbanken gespeichert und mit Datenbankmanagementsystemen (DBMS) verwaltet. Die Technologien hierzu sind bereits hoch entwickelt und weit verbreitet. Möchte man XML als Austauschformat für Daten unterschiedlicher Herkunft nutzen und gleichzeitig auf die Vorteile eines DBMS und die Vielzahl der Informationen, die in Datenbanken gespeichert sind, nicht verzichten, stellt sich die Frage, wie XML und DBMS sinnvoll kombiniert werden können und welchen zusätzlichen Nutzen sie einander bieten.

Die Vorteile des Einsatzes von Datenbanken zur Speicherung von XML-Daten gegenüber dem Dateienansatz sind vielseitig:

Effiziente und strukturierte Speicherung Datenbanken speichern Informationen in strukturierten Einheiten und bieten darüber hinaus unterschiedliche logische Sichten, ohne, dass der Anwender etwas über die Speicherung wissen muss.

Indexe Durch gezielte Indizierung der Daten lassen sich Informationen schneller auffinden.

Schnelle Anfrageauswertung Anfragen werden durch das DBMS optimiert, um schnellere Ergebnisse zu erhalten.

Sicherheitskonzepte Um die Datenintegrität zu gewährleisten, kann die Manipulation der Daten innerhalb gekapselter Transaktionen durchgeführt werden.

Mehrbenutzerfähigkeit Die meisten DBMS unterstützen den gleichzeitigen Zugriff durch mehrere Benutzer.

Standards DBMS sind weit verbreitet und viele Anwendungen verwenden bereits SQL-Schnittstellen (bzw. JDBC, ODBC o.A.) zum Zugriff auf Datenbanken.

XML-Dokumente können wegen ihrer speziellen Auszeichnungssyntax gezielt nach Informationen durchsucht werden. Eigens für XML entwickelte Anfragesprachen existieren bereits. Sind Informationen in Datenbanken gespeichert, können sie über SQL-Ausdrücke verarbeitet

und durchsucht werden, um z.B. dynamische Dokumente gezielt aus XML-Fragmenten zusammenzusetzen. Umgekehrt können große Datenmengen in XML durch Zerlegung in Ihre Elemente auf die Strukturen einer Datenbank abgebildet werden, um so schneller durchsucht werden zu können (Modelle für solche Abbildungen werden in Abschnitt 3.2 erläutert). Es bestehen dabei je nach Strukturierungsgrad und Anwendungsgebiet der Daten unterschiedliche Anforderungen an die Komplexität und die Verarbeitungsfunktionalität der Software bzw. des Datenbankmanagementsystems.

3.1 Produktkategorien

Die Einteilung der erhältlichen Systeme kann nach unterschiedlichen Kriterien erfolgen und einzelne Produkte sind nicht immer eindeutig zuzuordnen. Dies liegt an den komplexen Einsatzbereichen sowie an nicht eindeutig definierten technologischen Merkmalen. Die hier vorgestellte Einteilung und die Produktbeispiele sind der Arbeit von Ronald Bourret entnommen [Bou00b].

Softwaresysteme, die die XML-Welt mit der Datenbankwelt verbinden möchten, verfolgen unterschiedliche Lösungsansätze und bieten unterschiedliche Funktionen an. Die Einsatzbereiche umfassen den klassischen Austausch von Business-Data (B2B) durch Middleware, wobei meistens hoch strukturierte Daten verarbeitet werden müssen, bis hin zum Content-Management und damit zum Umgang mit verhältnismäßig wenig strukturierten Daten, also vom Menschen leicht lesbarer Information. Die Ansätze, mit denen diese Aufgaben bewältigt werden sollen, unterscheiden sich ebenso und reichen von der sogenannten *nativen Speicherung* von XML-Dokumenten über Struktur-Abbildungen zwischen XML-Dokumenten und (objekt-)relationalen Schemata bis zu Speicherung serialisierter DOM-Objekte.

3.1.1 Middleware

Middleware ist keine eigenständige Software. Sie muss auf unterschiedliche Art und Weise in neue Anwendungen integriert werden: Als Webserver Plug-In oder Add-On, als Java-Klassen, Perl-Module, Java-Beans oder Servlets. Middlewareprodukte werden verwendet, um Daten zwischen Datenbanken und anderen (Client-)Anwendungen zu transferieren. XML kann bei dieser Aufgabe als Austauschformat für die Daten dienen. Für den Datentransfer erzeugt die Client-Anwendung ein XML-Dokument mit den relevanten Daten, und ruft anschließend die Middleware auf, welche eine Datenbank kontaktiert, die XML-Daten für die Datenbank transformiert und schließlich in die Datenbank schreibt. Für das Überschreiben oder Löschen von Daten innerhalb der Datenbank werden häufig spezielle Tags innerhalb des XML-Dokuments verwendet (z.B. ADO von Microsoft).

Ein großer Teil der Middlewareprodukte kann lediglich Datenbankinhalte nach XML transferieren. Bei diesem Weg müssen in der Regel Anfragen in Form von Select-Statements durch die Client-Anwendung an die Middleware gestellt werden. Die Middleware liefert als Ergebnis ein XML-Dokument mit speziellen Tags oder eingebetteten Kommentaren. Die Struktur des Dokuments ist entweder durch das Datenbankschema vorgegeben oder kann durch den Aufbau des Select-Statements beeinflusst werden (z.B. XML SQL Utility for Java von Oracle). Einige Produkte verarbeiten anstelle von XML-Dokumenten auch DOM-Bäume (z.B. DatabaseDom von IBM).

Die meisten Erzeugnisse dieser Kategorie nutzen ODBC, JDBC oder OLE DB, um Zugriff auf eine Datenbank zu erhalten. Somit kann jede relationale Datenbank verwendet werden, die über einen entsprechenden Treiber verfügt.

3.1.2 XML-fähige Datenbanken

Zu der Gruppe der XML-fähigen Datenbanken zählen die üblicherweise relationalen DBMS mit speziellen Erweiterungen für die Verarbeitung von XML-Dokumenten und XML-formatierten Daten. Ihre zusätzlichen Funktionen ermöglichen die Speicherung und Rückgewinnung strukturierter XML-Dokumente typischerweise in Verbindung mit DTDs. Dabei wird die DTD für die Abbildung der hierarchischen Strukturen und der Einbettung der XML-Elemente und Attribute auf relationale Schemata herangezogen. Diese Abbildungen können bei einigen Systemen auch automatisch erfolgen, indem sie entweder aus einer gegebenen DTD ein (objekt-)relationales Schema generieren oder umgekehrt aus einem gegebenen Schema eine DTD für die zu verarbeitenden XML-Dokumente ableiten (z.B. Oracle8i). Genauso kann eine Abbildung zwischen den benutzerdefinierten Tabellen und XML-Elementen auch manuell vorgenommen werden (DB2 XML Extender von IBM).

Um XML-Dokumente aus der Datenbank zurück zu gewinnen stehen Hilfsprogramme, z.B. aus Basis von JDBC, zur Verfügung (XML SQL Utility for Java). In Ansätzen werden aber auch Erweiterungen von SQL durch spezielle XML-Funktionen eingesetzt. Die Speicherung von XML-Dokumenten kann aber auch vollständig in einer Spalte einer Tabelle erfolgen, beispielsweise als BLOB oder CLOB, wobei das Retrieval über textuelle Suchfunktionen erfolgen kann.

Zusätzlich verfügen die Systeme dieser Kategorie über Tools zur Unterstützung der Anwendungsentwicklung (z.B. Servlets, Klassenbibliotheken, XML-Parser usw.).

3.1.3 Native XML-Datenbanken

Eine große Anzahl von Datenbanksystemen werden von den Herstellern mit dem Schlagwort *native XML-Datenbank* beworben. Damit ist die ursprüngliche und unveränderte Speicherung und Rückgewinnung von XML-Dokumenten gemeint, wobei die Auslegung dieses Begriffes sehr offen gehandhabt wird. Grundsätzlich lassen sich vier Speichermethoden in diesem Bereich unterscheiden:

1. XML-Dokumente werden vollständig in Textform, in relationalen Datenbanken als BLOB, oder als Datei im Filesystem gespeichert. Dabei bleibt die ursprüngliche Struktur erhalten, allerdings ist die Performance bei Anfragen relativ gering.
2. Die Speicherung erfolgt in modifizierter Form, d.h. die Daten werden zuvor komprimiert oder binär in einer pre-parsed Form gespeichert.
3. Es wird eine strukturelle Abbildung von XML durchgeführt, wobei z.B. das Document Objekt Model (DOM) auf Tabellen oder direkt auf eine OO-Datenbank übertragen wird.
4. Die Struktur der Daten des XML-Dokuments, wie sie z.B. in einer DTD beschrieben ist, wird auf die Datenbank abgebildet. Beispielsweise wird für jeden Elementtyp eine

Tabelle im Schema der Datenbank angelegt. (Dieser Ansatz ist allerdings am wenigsten „nativ“, denn es werden im Prinzip nur die Daten gespeichert. Metadaten, wie Element- und Attribut-Namen werden auf Metaebene der Datenbank abgebildet und sind somit nicht durch einfaches Auslesen aus der Datenbank zu rekonstruieren.)

Native XML-Datenbanken verfolgen i.d.R. die Ansätze 2 und 3 (z.B. Tamino und LORE), wodurch die physikalische Struktur der Dokumente einerseits erhalten bleibt und andererseits auch XML-Dokumente gespeichert werden können, deren Struktur (DTD) vorher nicht bekannt ist. Datenbanken, die Option 4 umsetzen, lassen sich besser unter den XML-fähigen Datenbanken einordnen, da sie grundsätzlich ein angepasstes Datenschema erforderlich machen.

3.1.4 XML (Application-) Server

Die Grenze zwischen XML-Servern und XML-fähigen bzw. nativen XML-Datenbanken ist bei den vorhandenen Systemen teilweise fließend. XML-Server sind Datenserver für verteilte Anwendungen, die XML-Dokumente sowohl empfangen als auch bereitstellen können. Sie bieten sich oft als Plattformen für komplette E-Commerce Lösungen an und beinhalten vollständige Entwicklungsumgebungen für XML-basierte Anwendungen (z.B. eXcelon von Object Design/eXcelon Corp. sowie Tamino). XML-Server setzen sich meist aus Kerndiensten wie XML-Parser, XQL- und XSL-Interpreter, XML-Datenbank, Verwaltungsdiensten für Konfiguration und Benutzerverwaltung, Schnittstellen zu externen Datenquellen und individuellen Erweiterungen zusammen.

Reine XML Application Server können XML-Dokumente lediglich über das Internet versenden. Sie klinken sich beispielsweise in einen vorhandenen Web-Server wie Apache ein und fangen Anfragen für XML-Dokumente ab, können diese dann dynamisch aus einer Datenbank erzeugen oder mit einem XSL-Stylesheet verarbeiten. Die Generierung von Dokumenten erfolgt meist über vorgefertigte Templates mit eingebetteten SQL-Statements wobei der Server Parameter ggf. über HTML-Formulare erhalten kann. Die Verarbeitung wird z.B. über Perl-Skripte oder eXtensible Server Pages XSP (Cocoon von Apache.org) gesteuert.

3.1.5 Content Management Systeme

Content Management Systeme auf der Basis von XML unterstützen Speicherung, Retrieval und Generierung von Dokumenten aus mehreren Fragmenten. Die Systeme müssen im Gegensatz zu den Application Servern überwiegend mit weniger strukturierten XML-Dokumenten umgehen, also Dokumente, die größere Textanteile aber z.B. auch Grafiken enthalten. Sie bieten deshalb auch für die Dokumentbearbeitung typische Funktionen wie Editoren, Versionskontrolle, Mehrbenutzerfähigkeit und Volltextsuche. Die Anbindung an die Datenbank wird normalerweise vor dem Benutzer verborgen und spielt somit nur eine untergeordnete Rolle.

Viele Content Management Systeme verfügen zusätzlich über Verbreitungsmöglichkeiten über das Web und zusätzlich zu XML beherrschen sie ebenfalls SGML und verschiedene Grafik-, Audio- und Videoformate um komplexe Informationsquellen integrieren zu können (z.B. POET Content Management Suite von Sörman Information AB [Sör00]).

3.1.6 Persistente DOM Implementierungen

Das *Document Object Model* (DOM) bietet als API die Möglichkeit einer objektorientierten Repräsentation von XML-Dokumenten in einer baumartigen Datenstruktur [Hég00]. Die Darstellung von großen XML-Dokumenten als DOM-Bäume ist allerdings sehr speicherintensiv und die Konvertierung kostet viel Zeit. Deshalb nutzen Persistente DOM Implementierungen objektorientierte oder objektrelationale Datenbanken zum Speichern von XML-Dokumenten in Form von DOM-Objekten. Wie bei nativen XML Datenbanken, die zum Teil ebenfalls dieses Prinzip verwenden, bleibt dabei die Struktur der Dokumente erhalten. Im Vergleich zu nativen XML Datenbanken fehlen den Anwendungen dieser Kategorie aber beispielsweise Möglichkeiten, über eine URL Zugriff auf die Datenbank zu erhalten oder externe Datenquellen einbinden zu können. Persistente DOM Implementierungen eignen sich gut für die Nutzung in DOM-basierten Anwendungen, wenn eine Speicherung der Daten gewünscht wird.

3.2 Abbildungen zwischen XML und Datenbanken

Bei der Auswahl eines Datenbankmanagementsystems zur Speicherung von XML-formatierten Informationen spielen die Struktur und der Inhalt der XML-Dokumente eine wesentliche Rolle. Es können grundsätzlich zwei Klassen von Dokumenten unterschieden werden:

- Daten-spezifische Dokumente und
- Dokument-spezifische Dokumente

Daten-spezifische Dokumente sind für die Informationsverarbeitung und den Einsatz klassischer Datenbanktechnologien traditionell von größerer Bedeutung. Sie zeichnen sich durch einen hohen Strukturierungsgrad, sehr stark zerteilte informationstragende Einheiten und wenig natürlichsprachliche Inhalte aus. Dadurch können sie leicht maschinell generiert und verarbeitet werden. In diese Klasse fallen z.B. Daten des elektronischen Zahlungsverkehrs, Kataloge und alle Arten von Formulardaten. Liegen die Informationen als XML-Dokument vor, gibt es geeignete Methoden die relevanten Informationen zu extrahieren und in klassischen relationalen oder objekt-relationalen Datenbanken zu speichern.

Bei Dokument-spezifischen Dokumenten sind weniger einzelne Daten und Informationen als das Dokument als Ganzes von Interesse. Die Darstellung der Informationen in dieser Klasse von Dokumenten richtet sich mehr an den menschlichen Leser, ist deshalb weniger strukturiert aber dafür in einen stärker zusammenhängenden Kontext gefasst. In der Regel werden in solchen Dokumenten nur Metadaten, wie z.B. Autor, Erscheinungsjahr, Überschriften etc., durch spezielle Zusatzinformationen (*Tags*) hervorgehoben, die für eine weitere maschinelle Verarbeitung (z.B. automatische Katalogisierung und maschinelle Suche) von Interesse sind. Für die Verwaltung dieser Dokumente eignen sich am besten Content Management Systeme wie sie im vorangegangenen Abschnitt beschrieben wurden.

Dieser Abschnitt befasst sich vorrangig mit der Frage, wie Daten-spezifische XML-Dokumente und klassische Datenbankstrukturen zusammengeführt werden können. Dabei geht es darum, geeignete Abbildungen und Transformationsmechanismen zwischen beiden „Informationsträgern“ zu finden. Nach [Bou00a] lassen sich in ihrer Komplexität und Flexibilität sehr unterschiedliche sogenannte *Mapping*-Verfahren unterscheiden, wobei zwei grundlegend unterschiedliche Ansätze verfolgt werden: *Template-driven* und *Model-driven*.

3.2.1 Template-Driven Mapping

Das Template-driven Mapping ermöglicht lediglich den Transfer von Informationen in eine Richtung: Datenbankinhalte können nach XML abgebildet werden. In Verbindung mit XML-Stylesheets stellt dieses Modell aber beispielsweise ein sehr einfaches Verfahren zur dynamischen Erzeugung von strukturierten Webseiten dar.

Beim Template-driven Mapping existiert keine fest definierte Abbildung (Mapping) zwischen der Struktur des XML-Dokuments und der Datenbankstruktur bzw. dem Datenbankschema. Stattdessen werden XML-Dokumente verwendet, die als Templates (Schablonen) dienen sollen. Sie enthalten, wie in Abb. 1 dargestellt, spezielle eingebettete Anweisungen z.B. als SELECT-Statements. Diese Anweisungen erscheinen selbst als XML-Elemente mit eigens definierten XML-Tags. Die Darstellungsweise hängt dabei von dem verwendeten Programm ab, welches das Template später verarbeitet. Templates werden also eingesetzt, um gezielt spezielle Informationen, z.B. aus einer SQL-Datenbank in ein XML-Dokument zu übertragen.

```
<?xml version="1.0"?>
<FlightInfo>
  <Intro>
    the following flights have available seats:
  </Intro>
  <SelectStmt>
    SELECT Airline, FltNumber, Depart, Arrive
    FROM Flights
  </SelectStmt>
</FlightInfo>
```

Abbildung 1: XML-Template mit Select-Anweisung

Das Template wird beispielsweise von einem Server oder einer Transfer-Middleware verarbeitet. Dazu wird nach dem Parsen die Select-Anweisung herausgefiltert, an die Datenbank weitergeleitet und anschließend das Anfrageergebnis in XML formatiert und anstelle des Elements (im Beispiel `<SelectStmt>`) in das Dokument eingesetzt. Das Ergebnis kann dann wie in Abb. 2 aussehen.

Die Anwendung des Template-driven Mappings ist sehr flexibel, da die Formatierung des Anfrageergebnisses durch Erweiterung und Verschachtelung der Select-Anweisungen beeinflussbar ist. Einige Produkte unterstützen auch parametrisierte Templates und ermöglichen eine freie Platzierung der Anfrageergebnisse sowie den Einsatz von programmiersprachlichen Konstrukten wie Schleifen und Bedingungsabfragen.

3.2.2 Model-Driven Mappings

Model-driven Mappings ermöglichen eine Abbildung sowohl von einer Datenbank nach XML als auch umgekehrt von XML in eine Datenbank. Im Gegensatz zum Template-driven Mapping wird dafür bei den Model-driven Mappings ein festes Modell für die Struktur bzw. die Daten in jedem XML-Dokument vorausgesetzt, das auf die Datenbank abgebildet werden soll. Gleiches gilt für die Abbildung in umgekehrter Richtung. Es gibt zwei verbreitete Modelle, die in den

```

<?xml version="1.0"?>
<FlightInfo>
  <Intro>
    the following flights have available seats:
  </Intro>
  <Flights>
    <Row>
      <Airline>ACME</Airline>
      <FltNumber>123</FltNumber>
      <Depart>Dec 12, 2001 13:43</Depart>
      <Arrive>Dec 13, 2001 01:21</Arrive>
    </Row>
    ...
  </Flights>
</FlightInfo>

```

Abbildung 2: XML-Template Beispiel mit Ergebnis

meisten Systemen für den Austausch von Daten zwischen XML und Datenbank verwendet werden: Das *Table Model* und das *Data Specific Object Model*.

Das Table Model

Das Table Model ist wenig flexibel, bietet aber eine sehr intuitive Abbildung zwischen einem XML-Dokument und den Tabellen einer relationalen Datenbank. Abgebildet werden mit diesem Modell allerdings nur sehr flache XML-Dokumente, mit bis zu einer Tiefe von 3 eingebetteten Elementen. Aus diesem Grund ist der Einsatzbereich dieses Modells auf relationale Strukturen beschränkt, und ein XML-Dokument kann lediglich als Zwischenspeicher für die relationalen Daten angesehen werden. Das folgende Beispiel zeigt die Zuordnung der Struktur möglicher XML-Dokumente zu den Elementen der Datenbank:

```

<database>
  <table>
    <row>
      <column1>...</column1>
      <column2>...</column2>
      ...
    </row>
    ...
  </table>
  ...
</database>

```

Für den Fall, dass nicht der vollständige Tabelleninhalt nach XML transferiert werden soll, kann der Inhalt des *table-Tags* auch als Ergebnis einer Anfrage (*resultset*) interpretiert werden. Sind in dem XML-Dokument nur unvollständige Datensätze, also nicht alle Spalten einer Tabelle enthalten, handelt es sich um eine spezielle Sicht (*view*) auf die Datenbank. Die Abbildung

der XML-Elemente erfolgt also auf Meta-Elemente der Datenbank, wie Tabellen, Spalten und Views. Die Zuordnung der XML-Elemente zu den entsprechenden Elementen der Datenbank geschieht meist über ihre konkreten Namen oder über zusätzlich vereinbarte Attribute.

Die Vorteile dieses Modells sind einerseits seine Einfachheit und die damit verbundene leichte Implementierbarkeit, andererseits gelingt aber auch ein schneller und effizienter Datenaustausch zwischen mehreren relationalen Datenbanken. Bei komplexeren Datenmodellen und tiefer verschachtelten XML-Dokumenten ist es überhaupt nicht anwendbar.

Das Data-Specific Object Model

Während das Table Model mit einem Modell der allgemeinen Struktur von Datenbank und XML-Dokument arbeitet, werden beim Data-Specific Object Model die Daten selbst modelliert. Sie werden auf korrespondierende Objekte abgebildet, die wiederum zu hierarchischen Baumstrukturen, wie bei verschachtelten Elementen in einem XML-Dokument, angeordnet werden können. Auf diese Weise kann z.B. aus dem folgenden XML-Fragment ein sogenannter datenspezifischer Objektbaum, wie in Abb. 3, generiert werden.

```
<Orders>
  <SalesOrder SONumber="12345">
    <Customer CustNumber="543">
      ...
    </Customer>
    <OrderDate>150999</OrderDate>
    <Line LineNumber="1">
      <Product Name="Cherries">
        ...
      </Product>
      <Quantity Unit="ton">2</Quantity>
    </Line>
  </SalesOrder>
</Orders>
```

Bei dieser Abbildung werden Elemente zu Objekten und Subelemente sowie Attribute werden zu Eigenschaften von Objekten. Das so entstandene Modell kann dann direkt auf eine objektorientierte oder eine hierarchische Datenbank abgebildet werden.

Bei einem solchen Objekt-Mapping ist es prinzipiell nicht notwendig, dass jedem Element ein eigenes Objekt zugeordnet wird. Stattdessen werden zwei Arten von Elementen unterschieden: *Komplexe Elemente* sind solche, die Subelemente und Attribute besitzen (<Line LineNumber="1">), einfache Elemente enthalten lediglich PCDATA (<OrderDate>). Einfache Elemente können somit zu einer Eigenschaft (Property) ihres Elternelements bzw. -Objekts werden und komplexe Subelemente werden, wie in Abb. 4, durch Zeiger referenziert.

Sollen die Daten auf eine relationale Datenbank abgebildet werden, ist schließlich noch ein objekt-relacionales Mapping durchzuführen. Dabei kann entweder eine eindeutige Eigenschaft eines Objekts als Primärschlüssel dienen, oder ein neues Attribut wird speziell zu diesem Zweck definiert. Im oberen Beispiel wird also die Eigenschaft `soNumber` in der zugehörigen Relation `SalesOrder` als Primärschlüssel definiert und in der entsprechenden Relation des Subelements `Line` als Fremdschlüssel aufgenommen.

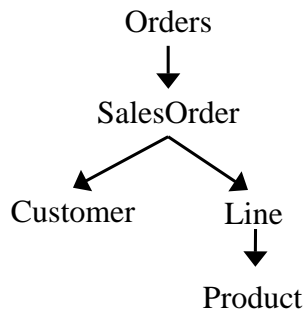


Abbildung 3: Daten-spezifischer Objektbaum

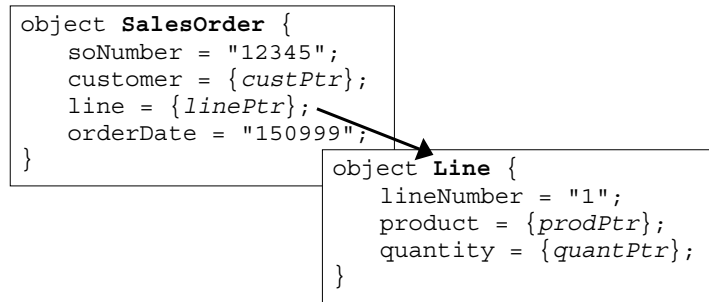


Abbildung 4: Beispiel für abgeleitete Objekte

SalesOrder	
<i>soNumber</i>	<i>orderDate</i>
12345	150999

—

Line	
<i>lineNumber</i>	<i>soNumber</i>
1	12345

Gegenüber dem Table Model hat dieses Verfahren den Vorteil, dass nahezu alle Arten von XML-Dokumenten verarbeitet werden können. Da die Daten und nicht die Struktur des Dokuments modelliert werden, ist die Verschachtelungstiefe der Elemente nicht beschränkt. Beim Transfer von Daten aus einer Datenbank können ebenfalls komplex strukturierte XML-Dokumente erzeugt werden. Dabei bleibt die logische Struktur der Daten für jede Verarbeitungsrichtung erhalten. Aus diesem Grund wird das Prinzip dieses Objekt-Modells in vielen Datenbanksystemen für XML genutzt, z.B. IBM DB2, Oracle8i, Informix, Microsoft SQL Server, XML-DBMS.

Beim Einsatz von relationalen Datenbanken ist es wichtig, dass die XML-Dokumente, die verarbeitet werden sollen, mit dem verwendeten Datenbankschema zusammenpassen, damit ein geeignetes Mapping möglich ist. Diese Tatsache macht auch deutlich, dass bei einem gegebenen Datenbankschema nicht jedes beliebige XML-Dokument nach dem Objekt-Modell generiert oder gespeichert werden kann. Für die meisten kommerziellen Bereiche sind die Daten und Informationen, mit denen umgegangen wird, allerdings fest definiert. Die Definition steckt entweder bereits in bestehenden (relationalen) Datenbankschemata oder ist im Fall von XML innerhalb vereinbarter DTDs kodiert.

3.2.3 DTDs und Datenbankschemata

Sollen bestimmte Typen von XML-Dokumenten, also beispielsweise solche mit einer spezifizierten DTD, in einer relationalen Datenbank gespeichert werden, kann die automatische Generierung eines relationalen Schemas für viele Anwendungen sehr hilfreich sein. Das folgende allgemeine Verfahren (nach [Bou00a]) betrachtet nacheinander Element- und Attributdefinitionen in einer DTD und baut aus den Informationen ein relationales Datenschema auf:

1. Für jedes komplexe Element eine Tabelle mit Primärschlüssel erzeugen.
2. Jeden Element-Inhalt (Subelemente und PCDATA) betrachten und
 - für einzelne Referenzen zu einfachen Elementen eine Spalte erzeugen,
 - für mehrfache Elemente (*) eine abhängige Tabelle mit Fremdschlüssel erzeugen,
 - für Referenzen zu komplexen Elementen einen Fremdschlüssel bei diesen erzeugen,
 - für PCDATA in komplexen Elementen eine abhängige Tabelle mit Fremdschlüssel in dieser erzeugen,
 - bei optionalen Elementen (?) Nullwerte in der entsprechenden Spalte erlauben und evtl. Spalten für Ordnungsnummern bei Subelementen und PCDATA einfügen, falls deren Reihenfolge wichtig ist.
3. Jedes Attribut betrachten und
 - für einfache Attribute eine Spalte erzeugen,
 - für mehrwertige Attribute (z.B. IDREFS) eine abhängige Tabelle mit Fremdschlüssel erzeugen,
 - bei optionalen Attributen (#IMPLIED) Nullwerte in der entsprechenden Spalte erlauben sowie Default-Werte übernehmen.

Umgekehrt kann aus einem gegebenen relationalen Schema eine DTD generiert werden:

1. Für jede Tabelle ein Element erzeugen, dessen Subelemente aus den Spalten der Tabelle abgeleitet werden. Gleichzeitig für jede Spalte, die nicht Primär- oder Fremdschlüssel ist, ein PCDATA-Element erzeugen.
2. Für jede abhängige Tabelle, die einen Fremdschlüssel enthält, ein Subelement erzeugen.
3. Bei Spalten mit möglichen Nullwerten ein optionales Subelement (?) erzeugen.

Die hier vorgeschlagenen Methoden unterliegen natürlich einigen zusätzlichen Einschränkungen. Eine DTD und ein relationales Datenschema haben zwar beide den Zweck Datenmodelle und Strukturen zu beschreiben, eine DTD lässt dabei aber wesentlich weniger Einschränkungen zu. Insbesondere können keine Datentypen und Längenbeschränkungen angegeben werden, was bei einer Schemadefinition besonders wichtig ist, um beispielsweise Datenintegrität und schnelle Berechnungen zu gewährleisten. Umgekehrt spiegelt sich in einem Datenschema keinerlei Reihenfolge bei der Abhängigkeit von Tabellen wider, die Reihenfolge der unterschiedlichen Subelemente muss in einem XML-Dokument aber unbedingt, so wie in seiner

DTD definiert, eingehalten werden. Andere Probleme sind z.B. mögliche Namenskollisionen, wenn in unterschiedlichen Tabellen gleichnamige Spalten auftauchen. In diesem Fall müssen die korrespondierenden Elemente in der DTD anders benannt werden. Für die meisten DTDs und Schemata gilt auch, dass die Übersetzungsmethoden nicht „Round-Trip“-fähig sind, d.h. ein aus einer DTD generiertes Schema lässt sich nicht in die ursprüngliche DTD zurückwandeln und umgekehrt. Trotzdem bietet beispielsweise Oracle⁸ⁱ die Möglichkeit aus einem gegebenen Schema oder einer Menge von Tabellen automatisch eine DTD zu generieren. Referenzen werden dabei sogar auf entsprechende ID- und IDREF-Attribute abgebildet [Wai99]. Unabhängig von den genannten Nachteilen sind die Probleme der vorgestellten Mapping-Modelle mit Zusatzinformationen, wie CDATA-Abschnitte, Kommentare, Entity-Referenzen und XML-Instruktionen, wenn diese in dem XML-Dokument erhalten bleiben sollen, da sie nicht zu den eigentlichen Nutzdaten zählen. Gleiches gilt für Metadaten, wie Dateinamen, DTDs etc.

3.2.4 Alternative Abbildungen

Zu den vorgestellten Mapping-Modellen existieren eine Reihe von alternativen Vorschlägen. Eine Möglichkeit ist z.B. die Speicherung von Struktur und Daten eines XML-Dokuments auf Instanzebene einer relationalen Datenbank [FK99]. Hierbei werden, etwas vereinfacht dargestellt, einzelne XML-Elemente einschließlich ihres Namens und PCDATA als zusammengehöriger Datensatz in einer Tabelle wie in Abb. 5 gespeichert. Als Zusatzinformationen werden eine ID, eine Eltern-ID sowie eine Ordnungszahl in derselben Tabelle aufgenommen. Attribute werden in einer ähnlichen Tabelle mit einer Referenz auf Ihr Elternelement gespeichert. Der Vorteil

Element	ID	Parent	Child#	String
buch	100	-		
titel	101	100	1	The XML companion
autor	102	100	2	
name	103	102	1	Bradley
vorname	104	102	2	Neil

Abbildung 5: Tabelle mit XML-Elementen auf Instanzebene

dieser Methode ist, dass nur wenige Tabellen mit fester Struktur benötigt werden. So lassen sich verschiedene XML-Dokumente, auch solche mit irregulärer Struktur und ohne DTD, leicht speichern und rekonstruieren. Die Nachteile zeigen sich bei der uneffizienten Realisierung von Abfragen, da sehr komplexe Joins notwendig sind, sobald mehr als ein Element oder Attribut auftritt.

Ein aufwändigerer Ansatz zur „natürlichen“ Speicherung von XML in relationalen Datenbanken, ähnlich dem Data-specific Object Model, wird in [STH⁺99] behandelt. Bei den sogenannten *Inlining Techniken* wird eine gegebene DTD zunächst sukzessive vereinfacht und die anschließende Darstellung als DTD Graph bildet die Grundlage zur Übersetzung in ein relationales Schema. Das *Inlining* besteht darin, verschachtelte Elemente ausgehend von einem möglichen Wurzelement in eine einzige Relation einzureihen. Dabei werden die tief gehenden Hierarchien der XML-Strukturen in relativ flache Relationen umgewandelt, wodurch einfache Anfragen sehr effizient durchgeführt werden können. Da in einer DTD aber kein Wurzelement

ausgewiesen wird, muss für jedes Element eine solche Relation aufgebaut werden. Der Nachteil der so erzeugten Redundanz kann durch leicht abgewandelte Techniken teilweise reduziert werden.

Häufig wird dem Anwender, der ein DBMS zur Speicherung von XML-Dokumenten und XML-formatierten Daten einsetzen möchte, die Möglichkeit gegeben, das Mapping der Daten vollständig manuell festzulegen. Sei es durch die Programmierung von Templates oder durch die teilweise auch grafisch unterstützte Erstellung sogenannter *Data-Access-Definitions* (bei IBM DB₂).

4 Beispiele für XML verarbeitende Datenbankmanagementsysteme

4.1 XML-fähige Datenbanken am Beispiel Oracle8i

Das Datenbanksystem Oracle8i bietet mit dem *Oracle XML Developer's Kit (XDK)* eine Vielzahl von Tools und Utilities für die Verarbeitung von XML [Wai99]. Das XDK besteht aus den folgenden Komponenten:

XML Parser und XSL Processor für Java, C, C++ und PL/SQL: Der XML Parser soll den Benutzer unterstützen, XML-Dokumente zu parsen und implementiert sowohl DOM als auch SAX Interfaces. Der XSL Transformation Processor ermöglicht die Transformation von XML in beliebige Textformate basierend auf XSL Style Sheets.

XML Class Generator für Java und C++: Der XML Class Generator für Java erzeugt Java Quelldateien basierend auf einer DTD. Die einzelnen Java Klassen stehen für die Elemente der DTD und können in Anwendungen verwendet werden, um Daten direkt zu manipulieren und neue XML-Dokumente zu generieren.

XML Transviewer Java Beans: Die Java Beans stellen einen DOM-Parser und unterschiedliche Viewer Funktionen in der grafischen Java Entwicklungsumgebung zu Verfügung.

Als Werkzeug für die Speicherung und Generierung von XML bietet Oracle das *XML SQL Utility for Java*. Mit Hilfe dieser Java Klassenbibliothek lassen sich auf vielfältige Weise Daten zwischen Datenbank und XML-Dokumenten austauschen. XML-Dokumente speichert Oracle entweder als vollständige Objekte vom Typ CLOB oder BLOB, oder in Relationen und Attributen dieser Relationen. Eine Kombination von beiden Modellen ist über speziell zu definierende Views ebenfalls möglich. Das XML SQL Utility verwendet für das Mapping auf (objekt-)relationale Tabellen das oben beschriebene Objekt Modell. Um XML-Dokumente zu generieren, unterstützt das Utility spezielle XML-Queries und gibt die Ergebnisse als XML in Form von Text oder DOM-Objekten zurück. Beim Speichern von XML-Dokumenten bildet das Utility Elemente auf die Spalten einer Tabelle ab (einfache Elemente auf skalare Spalten, komplexe Elemente auf Objekte). XML-Dokumente können aber nur gespeichert werden, wenn sie sich auf das gegebene Datenschema abbilden lassen. Für die Überprüfung der Dokumente kann eine DTD herangezogen werden, die das XML SQL Utility aus dem Datenschema ableiten kann.

Ein Werkzeug, das für die Generierung dynamischer XML-Dokumente auf Basis von Templates geeignet ist, steht mit dem *XSQL Servlet* zur Verfügung. Als Java Servlet verarbeitet es XML-Anfragen über eine URL auf dem Server. Die URL enthält ein XSQL-File und gegebenenfalls Parameter. Das XSQL Servlet verwendet den XML Parser sowie das XML SQL Utility für die Anfrageverarbeitung und sendet das Ergebnis nach weiterer Formatierung mit dem XSLT Processor als XML, HTML o.a. an den Client bzw. Browser oder eine andere Anwendung.

4.2 XML Server am Beispiel Tamino

Der XML Server *Tamino* („Transaction Architecture for the Management of INternet Objects“) der Software AG ist eine komplette XML Plattform für Electronic Business [Sof00]. Der Server basiert auf einem nativen XML-DBMS, der sogenannten *XML Engine*. Die Architektur des Servers beschreibt Abb. 6.

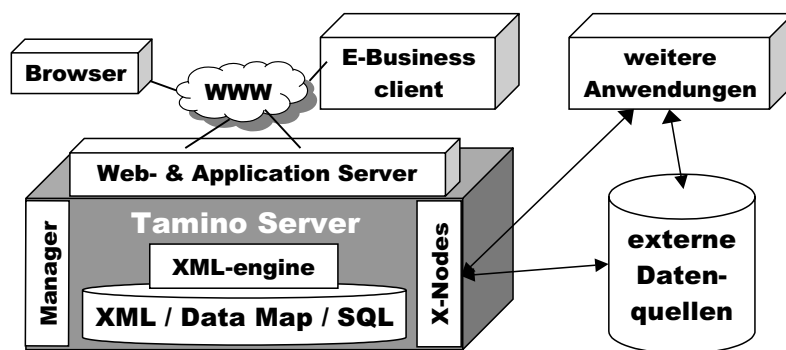


Abbildung 6: Architektur der Tamino XML Plattform

Die XML Engine speichert XML-Dokumente in ihrer ursprünglichen Form sowohl mit als auch ohne DTD. Andere Datenquellen (z.B. SQL Datenbanken oder Office Anwendungen) können über die *X-Nodes* zu einer einheitlichen Sicht zusammengeführt werden. Metadaten zu Struktur und Objekten von XML-Dokumenten sind in der *Data Map* gespeichert. Die Anwendungsentwicklung wird von Tamino durch das Software Developer's Kit (SDK) unterstützt, das auch eine Reihe von XML-Standardwerkzeugen enthält. Die Administration des Servers erfolgt über den *Manager*, dessen GUI über ein Java Applet auch dezentral aufgerufen werden kann.

Als Anfragesprache verwendet die XML Engine *X-Query*, einen auf dem XPath Standard [CD99] basierenden Dialekt. Anfragen können so als Teil einer URL gestellt werden. Das folgende einfache Beispiel liefert die Daten des Doktors eines Patienten mit Namen „Jones“:

...?_xql=hospital/patient[p-surname="Jones"]/doctor, wobei am Anfang die Adresse des Tamino XML Servers stehen muss. Wie die XML Engine eine Anfrage verarbeitet zeigt das Schema in Abb. 7. Eine Anfrage (X-Query) wird zunächst vom Query Interpreter anhand des in der Data Map gespeicherten Schemas analysiert. Der Object Composer konstruiert anschließend das entsprechende Ergebnis als XML-Dokument aus der Datenbank. Das Gegenstück zum Object Composer ist der Object Processor. Er ordnet die vom XML Parser

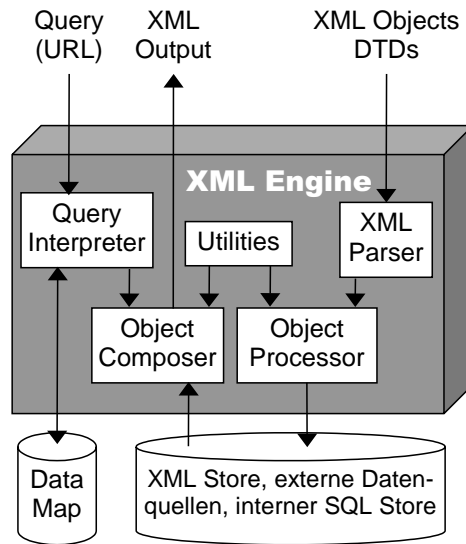


Abbildung 7: Architektur der XML Engine von Tamino

überprüften XML-Dokumente und Objekte bei der Speicherung entweder dem XML Store, der relationalen Datenbank oder einer externen Datenquelle zu.

5 Abschließende Anmerkungen

XML ist als Austausch- und Repräsentationsformat für strukturierte Daten zwischen Anwendungen, WWW und Datenbanken bereits weit verbreitet. Beim Einsatz von insbesondere relationalen Datenbanken zur Speicherung von XML müssen allerdings einige Einschränkungen in Kauf genommen werden. Der Aufwand für das Mapping der Daten darf nicht unterschätzt werden. Außerdem kommt es bei der Speicherung von statischen XML-Dokumenten in rein relationalen Tabellen zum Verlust von evtl. vorhandenen Metadaten und Zusatzinformationen. Bei unstrukturierten Dokumenten muss auf eine effiziente Speicherung der Elemente in Tabellen verzichtet werden. Diese Nachteile versuchen native XML-Datenbanken wie z.B. Tamino zu beseitigen, indem sie XML in ihrer ursprünglichen Form speichern. Dies bewirkt als weiteren Vorteil, dass der Aufwand XML- und Datenbankstrukturen einander anzupassen entfällt und eine aufwendige und zeitraubende Konvertierung von Anfrageergebnissen nicht notwendig ist.

XML-Datenbanken sind in der Lage ein einheitliche Sicht auf verteilte Daten unterschiedlicher Typen zu ermöglichen. Aufgrund neuer effizienter und einfacher Anfragesprachen für XML haben sie sich neben reinen OODBMS und RDBMS bereits in einigen Bereichen als neue Technologie etabliert.

Literatur

- [ABS00] Serge Abiteboul, Peter Buneman und Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufman Publishers, 2000.
- [Bos00] Bert Bos. Cascading Style Sheets. <http://www.w3.org/Style/CSS/>, Oktober 2000.
- [Bou00a] Ronald Bourret. XML and Databases. <http://rpbourret.com/XML/XMLandDatabases.htm>, November 2000.
- [Bou00b] Ronald Bourret. XML Database Products. <http://rpbourret.com/XML/XMLDatabaseProds.htm>, November 2000.
- [BPSM00] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen und Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation. <http://www.w3.org/TR/REC-xml/>, Oktober 2000.
- [CD99] James Clark und Steve DeRose. XML Path Language (XPath) Version 1.0, W3C Recommendation. <http://www.w3.org/TR/xpath>, November 1999.
- [CT00] Dan Connolly und Henry Thompson. XML Schema. <http://www.w3.org/XML/Schema/>, April 2000.
- [FK99] Daniela Florescu und Donald Kossmann. Storing and Querying XML Data using an RDBMS. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 22(3):27–34, September 1999.
- [Fro00] Max Froumentin. Extensible Stylesheet Language (XSL). <http://www.w3.org/Style/XSL/>, Oktober 2000.
- [Hég00] Philippe Le Hégarret. Document Object Model (DOM). <http://www.w3.org/DOM/>, Oktober 2000.
- [MKH00] Holger Meyer, Meike Klettke und Andreas Heuer. Datenbanken im WWW - Von CGI bis JDBC und XML. *HMD-Praxis der Wirtschaftsinformatik*, 214:5–22, November 2000. Erschienen im dpunkt.verlag, Heidelberg.
- [RT00] Claus Rautenstrauch und Klaus Turowski. XML-Datenbanken: Technische Grundlagen und betriebliche Anwendungen. *HMD-Praxis der Wirtschaftsinformatik*, 214:35–46, November 2000. Erschienen im dpunkt.verlag, Heidelberg.
- [Sof00] Software AG. Tamino - Technical Description. <http://www.softwareag.com/tamino/technical/description.htm>, 2000.
- [Sör00] Sörman Information AB. Content Management Suite. <http://www.sorman.se/products/cms/index.asp>, 2000. In October, 2000, Sorman Information AB bought POET Content Management Suite. It holds the intellectual property rights for CMS worldwide, except in the US, Canada, and South Korea, and has a six-month option to buy the rights in the US and Canada. (Quelle: Ronald Bourret: <http://rpbourret.com/XML/XMLDatabaseProds.htm>).

- [STH⁺99] Jayavel Shanmugasundaram, Kristin Tufte, Gang He, Chun Zhang, David DeWitt und Jeffrey Naughton. Relational Databases For Querying XML Documents: Limitations and Opportunities. In *Proceedings of the 25th VLDB Conference*, Edinburgh, Scotland, September 1999.
- [Wai99] Brad Wait. Using XML in Oracle Database Applications. <http://technet.oracle.com/tech/XML/info/>, November 1999.
Part 2: About Oracle's XML Products
Part 3: Customizing Data Presentation
Part 4: Exchanging Business Data Among Applications.