XML-Datenbanksysteme und ihre Anwendung

XML Database Systems and their Application

Sven Helmer, Carl-Christian Kanne, Guido Moerkotte, Universität Mannheim

Zusammenfassung Mit zunehmender Verbreitung von XML werden die in XML vorliegenden Datenbestände immer größer und die auf XML basierenden Anwendungen immer komplexer. Daher sind effektive und effiziente Methoden für die Verwaltung persistenter XML-Bestände gefordert. Wir betrachten die wichtigsten gemeinsamen Anforderungen an eine Persistenzlösung für XML und besprechen verschiedene Alternativen zur Speicherung. Es wird sich herausstellen, dass lediglich speziell für XML entwickelte Datenbankmanagementsysteme den Anforderungen genügen. Wir geben daher einen Überblick über ein exemplarisches XML-Bank-Managementsystem namens Natix. ►►►

Summary Due to its fast proliferation, the amount of XML data is steadily increasing and XML-based applications become more and more complex. This requires effective and efficient methods to manage large amounts of persistent XML. We identify the most common requirements for persistence solutions and discuss alternative approaches to store XML. Our conclusion will be that only database management systems specifically designed for XML fulfill the requirements. Last, we give an overview of one such system called Natix.

KEYWORDS database systems, XML

1 Einleitung

Schon wenige Jahre nach der Verabschiedung des XML-Standards durch das W3C erreicht XML eine hohe Akzeptanz in breiten Schichten sowohl der Softwarehersteller als auch der Anwender. Ein wesentlicher Faktor für diesen Erfolg liegt in der *hohen Flexibilität von XML*. Es eignet sich für die Beschreibung von Entitäten mit einem sehr unterschiedlichen Grad an Strukturiertheit. Auf der einen Seite des Spektrums liegen die *dokumentzentrierten* Entitäten. Hierbei handelt es sich um Dokumente, die vornehmlich für den menschlichen Leser bestimmt sind, wie die Werke von Shakespeare, die Bibel oder Marketingbroschüren. Auf der anderen Seite des Spektrums liegen stark strukturierte XML-Dokumente, die aus reinen Daten, versehen mit Metainformation, bestehen. Diese Verwendung von XML wird als *datenzentriert* bezeichnet. Beispiele in diesem Bereich sind Sternkataloge, Erdbebendaten, annotierte Gensequenzdaten oder auch Bestellungen. Dazwischen gibt es die semistrukturierten Dokumente. Man findet sie beispielsweise in Form von Geschäftsberichten, die neben Textbausteinen auch Daten in Reinform enthalten, wie beispielsweise Angaben zur Geschäftsführung und wichtige Kennzahlen eines Unternehmens. Wir finden semistrukturierte Entitäten aber auch in Form von Katalogeinträgen, die beispielsweise sowohl technische Daten eines Verstärkers enthalten als auch beschreibenden Text und Abbildungen. Mit zunehmender Verbreitung von XML stellt sich das Problem einer adäquaten Verwaltung der persistent zu haltenden XML-Dokumente. Es ist offensichtlich, dass durch die angestrebten Persistenzlösungen das gesamte Spektrum der Strukturiertheit abgedeckt werden muss.

Prinzipiell gibt es drei Möglichkeiten der persistenten Speicherung von XML: Speicherung in einem Dateisystem, Speicherung in einer nicht-XML-Datenbank, wie einer (objekt-)relationalen oder auch objektorientierten Datenbank, und Speicherung in einer speziell für XML entwickelten Datenbank. Letztere werden auch *native* XML-Bank-Managementsysteme (XBMS) genannt.

Die meisten XML-Dokumente werden in Dateien gespeichert,

weil dieser Ansatz sehr einfach realisierbar ist und für kleinere Dokumentkollektionen eine durchaus adäquate Möglichkeit darstellt. Probleme ergeben sich hieraus erst, wenn die Dokumentkollektion wächst und höhere Ansprüche an die Anwendungen gestellt werden, die auf den XML-Dokumenten arbeiten.

Für ein oder mehrere XML-Dokumente sind vielfältige Anwendungen und Werkzeuge denkbar, die gegebene Problemstellungen lösen. Soll beispielsweise eine schnelle Schlüsselwortsuche realisiert werden, so muss ein Volltextindex aufgebaut werden. Dies erfolgt mit einem speziellen Werkzeug, und der Index selbst wird in einer eigenen Datei abgelegt. Hat man nun viele derartige Werkzeuge und zusätzliche Dateien, so führt dies zu einem Konglomerat von Dateien und Werkzeugen, das nur noch schwer zu handhaben ist.

Die meisten Werkzeuge bearbeiten XML-Dokumente über standardisierte Schnittstellen wie etwa DOM oder SAX. Eine DOM- oder SAX-Sicht auf ein Dokument wird dabei von einem Parser geliefert. Bild 1 zeigt die Situation für einige Standardanwendungen wie XML-Editor, XPath-Prozessor, XQuery-Prozessor sowie Volltextsuche auf XML-Dokumenten.

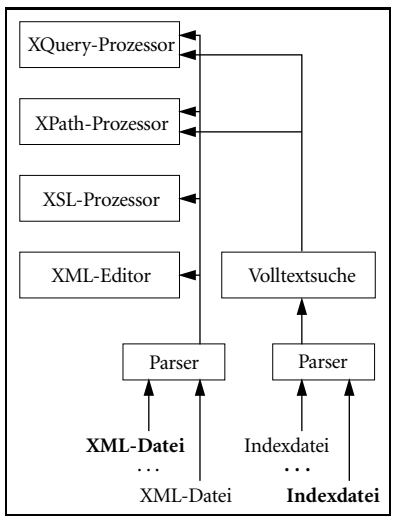


Bild 1 Speicherung von XML in Dateien.

Neben dem oben genannten Problem ergibt sich ein weiteres, wenn ein gewisser Durchsatz verlangt wird. Sind alle XML-Dokumente in Dateien abgelegt, so müssen sie von jedem Werkzeug vor ihrer Verarbeitung gelesen, syntaktisch analysiert, unter Umständen validiert und oft auch noch in eine Hauptspeicherrepräsentation umgewandelt werden. Dies bedeutet einen zusätzlichen Aufwand, der bei großen Anwendungen mit einer hohen Zahl von Benutzern nicht vertretbar ist. Hinzu kommt, dass die Unterstützung von Transaktionsverwaltung (Recovery und Synchronisation, d. h. Isolation verschiedener Anwendungen) in Dateisystemen recht rudimentär ist.

Im nächsten Abschnitt beschäftigen wir uns zunächst etwas detaillierter mit den Anforderungen an Datenbanksysteme zur persistenten Speicherung von XML. Diese Anforderungen leiten wir aus verschiedenartigen Anwendungsszenarien her. Im darauffolgenden Abschnitt zeigen wir die Probleme auf, die entstehen, wenn man – wie vielfach propagiert – ein (objekt-)relationales Datenbanksystem einsetzt. Die Schlussfolgerung dieses Abschnitts wird sein, dass nur Datenbanksysteme, die speziell für XML entwickelt wurden, so genannte *native* XBMSe, eine adäquate Lösung darstellen. In Abschnitt 4 stellen wir mit Natix ein solches System vor. Abschnitt 5 fasst die wesentlichen Aussagen des Artikels zusammen.

2 Anforderungen an eine Datenhaltung für XML

Um Anwendungsentwickler effektiv von den Details der Verwaltung und Anfragebeantwortung bei persistenten XML-Dokumenten zu entlasten, muss ein allgemeines Datenhaltungssystem folgende Punkte erfüllen

(1) Effiziente Speicherung von XML-Dokumenten und effizienter Zugriff auf die gespeicherten Dokumente und Dokumentteile.
(2) Unterstützung von standardisierten, deklarativen Anfragesprachen wie XPath und XQuery.
(3) Unterstützung von standardisierten Anwendungsprogrammierschnittstellen wie DOM und SAX.
(4) Eignung für eine Mehrbenutzerumgebung, insbesondere Bereitstellung einer Transaktionsverwaltung mit Synchronisation und Recovery.

Wir illustrieren diese Anforderungen mit zwei typischen Anwendungsszenarien.

In einem Online-Auktionshaus werden eine Vielzahl von Produkten in unterschiedlichen Kategorien angeboten. Je nach Kategorie gibt es unterschiedliche Attribute, bei Kraftfahrzeugen z. B. Motorleistung, Erstzulassung und Farbe, bei Software hingegen Version und Hersteller. Beschreibende Texte und Abbildungen vervollständigen die Produktbeschreibungen.

Für eine internetbasierte Lösung bietet sich die Beschreibung von Produkten mittels XML-Dokumenten an. Weitere XML-Dokumente fallen dann unter anderem in Form von redaktionellen Informationen, Beschreibungen von Auktionsteilnehmern, Schablonen für Webseiten und speziellen, themenbezogenen Produktzusammenstellungen an. Das Fortschreiten der Auktionen mit Geboten, Mitteilungen und schließlich Verkauf wird ebenfalls in Dokumenten festgehalten. Schneller Zugriff auf die Dokumente ist entscheidend, um viele gleichzeitig agierende Nutzer bedienen zu können (Punkt 1). Die Besucher und Redakteure des Auktionshauses benötigen komfortable Suchfunktionen, deren Implementierung durch deklarative Anfragesprachen erleichtert wird (Punkt 2). Als Beispiel diene die themenbezogene Produktzusammenstellung. Die Anbindung von Stylesheet-Prozessoren erfolgt üblicherweise über Standardschnittstellen (Punkt 3). Das Erstellen der redaktionellen Dokumente und

Schablonen erfolgt mit Editoren, die auf Standardschnittstellen wie SAX oder DOM arbeiten (Punkt 3). Die Vielzahl der gleichzeitig arbeitenden Akteure im System sowie die direkte wirtschaftliche Relevanz der verwalteten Dokumente macht eine zuverlässige Transaktionsverwaltung inklusive Mehrbenutzerbetrieb (Punkt 4) unverzichtbar.

Ein weiteres Beispiel für ein Anwendungsgebiet, das die genannten Anforderungen aufweist, sind die „Life Sciences“, wie z. B. Gentechnik, Molekularbiologie und Pharmaforschung. Dort werden vornehmlich mit Annotationen versehene Sequenzen verarbeitet, wie DNA, RNA oder Aminosäureketten. Solche Sequenzdaten werden mit einer Reihe von unterschiedlichen Werkzeugen bearbeitet, um die in Form von XML-Dokumenten abgelegten Sequenzen zu erzeugen, zu visualisieren, mit Zusatzinformationen zu annotieren und mit anderen Datenquellen zu konsolidieren (Punkt 1). Vielfach muss Software integriert werden, die auf XML-Standardschnittstellen wie DOM oder SAX aufbaut (Punkt 3). Die große Innovationsdichte im Bereich der Life Sciences resultiert in immer neuen Methoden, die schnell in Werkzeuge umgesetzt werden müssen. Deklarative Anfragesprachen beschleunigen die Anwendungsentwicklung durch eine extreme Vereinfachung von häufig wiederkehrenden Teilproblemen, wie der Wiederauffindung von Sequenzen und Sequenzteilen anhand von Annotationen (Punkt 2). Die Gewinnung neuer Erkenntnisse über Sequenzen erfordert kostspielige Experimente, die oft aus Zeitgründen parallel ausgeführt werden. Die experimentellen Ergebnisse finden als wertvolle neue Annotationen Eingang in die Datenbasis. Eine leistungsfähige Transaktionsverwaltung ist erforderlich, um Ressourcenverschwendung zu vermeiden (Punkt 4).

An skalierbare Datenhaltungssysteme für XML werden grundsätzlich ähnliche Anforderungen gestellt wie an Datenhaltungssysteme für andere Formen von Daten. Die typischen Schnittstellen, Anfragesprachen und zugrundeliegenden Datenmodelle sind jedoch sehr unterschiedlich. Im Folgenden begründen wir, warum relationale Datenbanksysteme die gestellten Anforderungen aus technischen Gründen nur mangelhaft erfüllen können.

3 Speicherung von XML in relationalen Datenbanksystemen

Zur Speicherung von XML-Dokumenten in relationalen Datenbanksystemen (RDBMS) gibt es viele verschiedene Ansätze [2–8]. Einige beschränken sich jedoch auf die Abbildung von XML-Elementen, mehrere betrachten noch XML-Attribute, darüber hinausgehende Konstrukte werden vernachlässigt. Der Bereich der möglichen Speicherungsformen von XML-Dokumenten in relationalen Datenbanksystemen wird abgesteckt durch die Speicherung von ganzen XML-Dokumenten als große Zeichenfelder, auch CLOB (*Character Large OBject*) genannt, auf der einen Seite und der Speicherung einzelner Dokumentknoten in einzelnen Tupeln auf der anderen Seite. Offensichtlich sind vielfältige Zwischenformen möglich.

Der einfachste Ansatz speichert jedes Dokument in einem CLOB. Wie Bild 2 zeigt, unterscheidet sich dieser Ansatz in den Auswirkungen nicht wesentlich von der Situation, die entsteht, wenn man direkt ein Dateisystem verwendet. Man hat jedoch den Vorteil des Transaktionsmanagements.

Der Export eines ganzen Dokuments ist schnell. Dagegen ist es sehr aufwändig, gezielt ein Fragment eines Dokuments zu exportieren. Hierzu muss der gesamte CLOB ausgelesen und syntaktisch analysiert werden, bevor das zu exportierende Fragment bestimmt werden kann.

Falls das RDBMS weitere Werkzeuge wie XSL-, XPath- und XQuery-Prozessor integriert, entfällt zumindest deren getrennte Verwaltung. Intern im RDBMS wird aber immer noch jedes in einem CLOB gespeicherte Dokument vor jeder Verarbeitung syntaktisch analysiert. Der hierfür zuständige Parser kann dann eine DOM- oder SAX-Schnittstelle zur Verfügung stellen, auf der auch andere Anwendungen arbeiten können. Dies führt zu entsprechenden Leistungsverlusten.

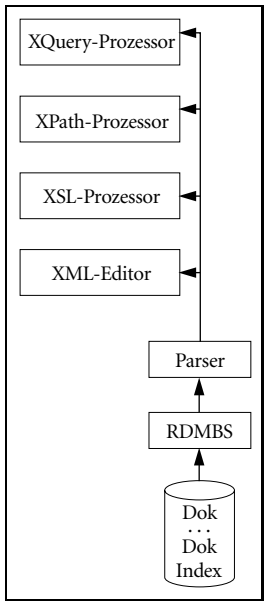


Bild 2 Anwendungen mit RDBMS, Dokumente als CLOB.

Wenn man jeweils einen Dokumentknoten in einem Tupel speichert, ist die Situation komplementär. Möglichkeiten, die Dokumentknoten auf Tupel abbilden, bewegen sich zwischen zwei Extremen. Zum einen kann man für XML-Dokumente ein generisches Schema erstellen, das dann für alle Dokumente unabhängig von der DTD genutzt werden kann, und zum anderen kann man für jede DTD ein Schema erstellen.

Es sind aber bei beiden Ansätzen die gleichen Kritikpunkte angebracht. Betrachten wir wieder, wie Anwendungen mit einer solchen Datenbank arbeiten können. Zunächst einmal stellen wir fest, dass der Import eines Dokumentes sehr aufwändig ist, da es in einzelne Knoten zerlegt werden muss, für die dann Tupel erzeugt werden müssen, die anschließend gespeichert werden. Das Exportieren eines gesamten Dokuments ist ebenfalls ein aufwändiges Unterfangen, da alle

Knoten des Dokuments zusammengesucht werden müssen. Erheblich verteuert wird das Exportieren von Dokumenten durch die Tatsache, dass Tupel in Relationen ungeordnet sind, die Elemente in XML-Dokumenten jedoch geordnet. Diese Ordnung gilt es beim Export zu rekonstruieren. Das Exportieren eines kleinen Dokumentfragmentes, wie der Inhalt eines einzelnen Textknotens, geht hingegen bei gegebenem Knotenidentifikator recht schnell.

Um weitergehende Anwendungen auf der Datenbank realisieren zu können, benötigt man wieder eine DOM- oder SAX-Schnittstelle. Bei der Speicherung von XML-Dokumentknoten in einzelnen Tupeln scheidet die Verwendung eines Parsers aus. Auf der anderen Seite können aber die Navigationsoperationen (andere selbstverständlich auch) mittels SQL nachgebildet werden. Den nächsten Geschwisterknoten findet die in Bild 3 dargestellte SQL-Anfrage. Man beachte den Aufwandsunterschied zwischen dieser Anfrage und der einfachen Dereferenzierung eines Zeigers. Wenn man sich jetzt kompliziertere DOM-Funktionen vorstellt, wie beispielsweise das Auffinden aller direkten und indirekten Nachfolger eines gegebenen Knotens, wird sehr schnell deutlich, wie aufwändig solche DOM-Operationen nachgebildet werden müssen. Oft muss eine rekursive Anfrage abgesetzt werden.

Bei der Verwendung von Anwendungen wie XPath- und XQuery-Auswertern hat man jetzt zwei Möglichkeiten. Zum einen können diese auf der DOM-Schnittstelle aufsetzen. Es sollte allerdings offensichtlich sein, dass sich dieses nach den vorherigen Betrachtungen aus Leistungsgesichtspunkten heraus verbietet. Die andere Möglichkeit besteht darin, XPath- und XQuery-Ausdrücke selbst direkt in SQL-Anfragen zu übersetzen.

```
select y.id
from  Dokumentknoten x, Dokumentknoten y
wherex.id = :id and x.vater = y.vater
      and x.kindnummer + 1 = y.kindnummer
```

Bild 3 DOM in SQL.

Hier ergeben sich drei Probleme. Zum einen bedeutet diese zusätzliche Abbildungsschicht von XPath bzw. XQuery nach SQL und für das Ergebnis von Relationen zu XML-Fragmenten/Knotenmengen einen Leistungsverlust. Zum zweiten ist es nicht möglich, alle XPath- oder XQuery-Ausdrücke in SQL zu übersetzen. Hierzu ist SQL nicht mächtig genug. Zum dritten sind die erzeugten SQL-Anfragen ähnlich ineffizient, wie für die DOM-Operationen demonstriert. Beispielsweise findet der simpelste XPath-Ausdruck „//“ alle direkten und indirekten Nachfolgerknoten eines gegebenen Knotens. Man kann sich also leicht vorstellen, wie aufwändig die Berechnung allgemeiner XPath-Ausdrücke wird. Von XQuery-Anfragen wollen wir hier lieber schweigen.

Statt dessen wenden wir uns noch einem weiteren kritischen Punkt zu. Speichert man Knoten verschiedener XML-Dokumente in einer Relation – und es ist davon auszugehen, dass man nicht für jedes Dokument neue Relationen anlegt –, so hat man bei der Synchronisation verschiedener Anwendungen folgendes Problem: Um Phantomprobleme zu vermeiden, also Knoten, die während der Bearbeitung mal sichtbar sind und mal nicht, muss in einem RDBMS immer die ganze Relation gesperrt werden, in der ein Tupel (Knoten) eingefügt wurde. Dies kann leicht dazu führen, dass mehrere Dokumente vollständig gesperrt werden, obwohl eine Transaktion vielleicht nur in einem Dokument einen Knoten einfügt. Dies ist eine oft unerträgliche Situation.

An dieser Stelle sei kurz angemerkt, dass die Verwendung eines OODBMS ähnliche Nachteile birgt, wie die Verwendung eines RDBMS bei Speicherung von Dokumentknoten in einzelnen Tupeln. Lediglich die Operationen der DOM-Schnittstelle können effizienter implementiert werden.

Aber nicht nur aus technischer, sondern auch aus betriebswirtschaftlicher Sicht ergeben sich einige Nachteile durch die Verwendung eines RDBMS zur Speicherung von XML-Dokumenten. Relationale Datenbankmanagementsysteme sind oftmals relativ alte, über Jahre und Jahrzehnte gewachsene Systeme, die für vielfältige Anwendungsbereiche mehrfach erweitert wurden. Daher handelt es sich um relativ komplexe, schwierig zu installierende und administrierende Systeme. Durch die notwendigen XML-spezifischen Erweiterungen verbessert sich diese Situation nicht. Verwendet man ein relationales DBMS nur für die Speicherung von XML-Dokumenten, so bezahlt man nicht nur Lizenzgebühren für viele Codezeilen, die man nie ausführen wird, man handelt sich auch den hohen Installations- und Administrationsaufwand ein.

Vorteilhaft bei der Verwendung eines RDBMS ist, dass oft bereits geschultes Personal in den Unternehmen vorhanden ist. Außerdem sind die RDBMS-Hersteller bemüht, die Administration ihrer Systeme zu vereinfachen.

Zum Abschluss sei noch angemerkt, dass wir bei zwei Verfechtern der Speicherung von XML in relationalen Datenbanksystemen, nämlich Florescu und Kossmann, folgenden Satz zur schlechten Leistung relationaler Systeme beim Export finden [2]:

> *These observations indicate that it might be advantageous to store copies of the original XML documents in the file system in addition to loading the XML data into an RDBMS.*

Die sich aus dieser nicht unter Datenbanksystemkontrolle stehenden Replikation ergebenden Probleme dürften offensichtlich sein. Unserer Meinung nach ist der Schluss zu ziehen, dass nur native XML-Datenbanksysteme eine adäquate Lösung für die Verwaltung von XML-Dokumenten im Rahmen komplexer Anwendungen darstellen.

4 Natix: Ein natives XBMS

Motiviert durch die dargestellten Nachteile bei der Verwendung von relationalen Systemen, wird am Lehrstuhl für Praktische Informatik III der Universität Mannheim seit Anfang 1998 ein natives Datenbanksystem für XML mit Namen Natix entwickelt [1]. Bei der Realisierung von Natix wurde Wert darauf gelegt, die oben geschilderten Probleme beim Einsatz von konventionellen DBMS zur Speicherung von XML an der Wurzel zu beheben. Durch die Berücksichtigung der XML-Anforderungen bei Entwurf und Implementierung aller Schichten des Systems wurde dabei ein sehr leistungsstarkes XBMS geschaffen. Es sei angemerkt, dass es neben Natix mit Tamino von der Software AG und anderen[1] noch weitere native XBMS gibt.

Die Architektur von Natix besteht aus drei Ebenen (Bild 4). Die unterste ist die Speicherebene, die für die Speicherung von XML-Daten, Metadaten und Indexstrukturen zuständig ist. In der nächsthöheren Ebene befinden sich die verschiedenen Datenbankdienste, die Anforderungen der Anwendungen erfüllen, die über einfache Speicherung hinausgehen.

Speicherebene und Dienste bilden zusammen die *Natix Engine*. Die Dienste kommunizieren untereinander und mit den Anwendungen durch das *Natix Engine Interface*, das eine einheitliche Schnittstelle für die gesamte Datenbank-Engine zur Verfügung stellt.

Auf Basis des Natix Engine Interface sind dann in der Anbindungsebene die verschiedenen Anwendungsanbindungen realisiert, in der, wenn nötig, die internen Repräsentationen von Dokumenten, Anfragen und Metadaten in die vom jeweiligen API benötigte Darstellung umgewandelt werden.

4.1 Speicherebene

Dieser Teil von Natix übernimmt die Verwaltung aller persistenten Daten. Dazu gehören neben den XML-Dokumenten auch Indexstrukturen, Metainformationen und Protokolle für die Recovery-Algorithmen der Transaktionsverwaltung.

4.2 Datenbankdienste

Das Natix Engine Interface stellt ein Framework zur Kommunikation der Dienste zur Verfügung. Operationen wie Compilation (Prepare) und Auswertung einer Anfrage, der Import und Export von Dokumenten werden als Request an das Engine Interface gestellt und an die jeweils beteiligten Dienste weitergeleitet.

Die virtuelle Maschine dient der Auswertung von Anfragen. Dabei kann sie direkt auf die Hintergrundspeicherrepräsentation der XML-Dokumente zugreifen, ohne diese zunächst mit hohem CPU-Aufwand in eine Hauptspeicherrepräsentation umzuwandeln.

Der *Query Compiler* übernimmt die Aufgabe, die in XML-Anfragesprachen formulierten Anfragen in Programme für die *Natix Virtual Machine* zu übersetzen.

Die Transaktionsverwaltung stellt die Infrastruktur für Mehrbenutzerbetrieb zur Verfügung, bei dem die ACID-Eigenschaften (*Atomicity*, *Consistency*, *Isolation*, *Durability*) für die parallel ablaufenden Transaktionen garantiert werden.

Dabei verlangt das Anforderungsprofil eines XBMS mit sehr vielen Teilobjekten pro Dokument auch spezielle Sperrtypen und Recovery-Protokolleinträge für Dokumente und Teildokumente. Eine konventionelle Transaktionskomponente ist hier schnell an der Grenze ihrer Leistungsfähigkeit.

Zur Vermeidung von wiederholten Repräsentationswechseln zwischen der Hintergrundspeicherrepräsentation und den verschiedenen, von den Anwendungen verwendeten Hauptspeicherrepräsentationen (z. B. DOM-Tree) von Dokumenten wird ein Objekt-Cache eingesetzt, auf den die verschiedenen Anwendungstreiber über einen Objektmanager zugreifen können.

[1] siehe http://pi3.informatik.uni-mannheim.de/~moer

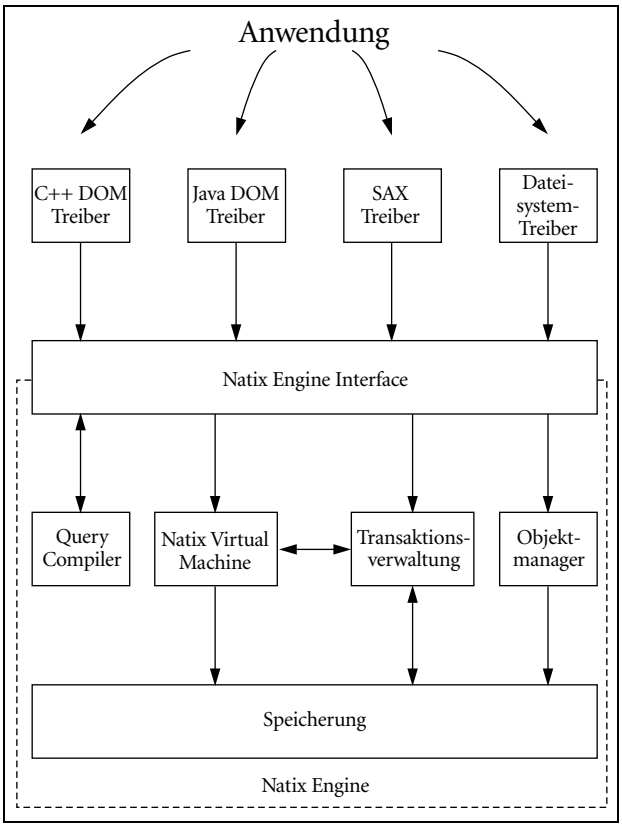


Bild 4 Architekturüberblick.

4.3 Anbindungsebene

In der Anbindungsebene sind die Schnittstellen zum Zugriff auf die Datenbank-Engine aus den verschiedenen Anwendungen realisiert.

Neben den obligatorischen DOM-Schnittstellen für C++ und Java ist hier auch ein Dateisystemtreiber angesiedelt, der den Datenbankinhalt allen Anwendungen zur Verfügung stellt, die mit regulären Dateien arbeiten können. Dadurch kann die eventuell bestehende anfängliche Hürde beim Einsatz eines neuen DBMS leichter genommen werden.

5 Zusammenfassung

Von den drei untersuchten Möglichkeiten der Speicherung von XML erfüllen nur XBMSe die erarbeiteten Anforderungen. Sowohl die Speicherung von XML in Dateien als auch die Verwendung von relationalen oder objektorientierten Datenbanksystemen erfüllen die Anforderungen nur mangelhaft.

Ungeachtet dieser Tatsachen wird oft das Argument der mangelnden Reife von XBMSen ins Feld geführt. Betrachtet man jedoch die Tatsache, dass diese bereits seit fünf Jahren entwickelt werden und vielfältig eingesetzt werden, so sieht man, dass dieses Argument zu kurz greift: Sowohl Natix als auch Tamino befinden sich bereits seit Jahren mit Erfolg im kommerziellen Einsatz.

Danksagung

Wir danken Simone Seeger für ihre Unterstützung bei der Erstellung des Manuskripts.

Literatur

[1] T. Fiebig, S. Helmer, C.-C. Kanne, G. Moerkotte, J. Neumann, R. Schiele, T. Westmann: Anatomy of a Native XML Base Management System. *VLDB Journal*, 11(4):292–314, 2002.

[2] D. Florescu, D. Kossmann; Storing and querying XML data using an RDBMS. *IEEE Data Engineering Bulletin*, 22(3):27–34, 1999.

[3] W. Kazakos, A. Schmidt, P. Tomczyk: *Datenbanken und XML*. Springer, 2002.

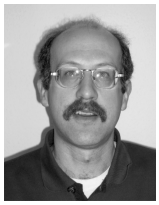
[4] M. Klettke, H. Meyer: XML and object-relational database systems – enhancing structural mappings based on statistics. In *ACM SIGMOD Workshop on the Web and Databases (WebDB)*, 2000.

[5] A. Schmidt, M. Kersten, M. Windhouwer, F. Waas: Efficient relational storage and retrieval of XML documents. In *ACM SIGMOD Workshop on the Web and Databases (WebDB)*, 2000.

[6] J. Shanmugasundaram, H. Gang, K. Tufte, C. Yhang, D. J. DeWitt, J. Naughton: Relational databases for querying xml documents: Limitations and opportunities. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 302–314, 1999.

[7] B. Surjanto, N. Ritter, H. Loeser: XML content management based on object-relational database technology. In *Proc. Int. Conf. on Web Information Systems Engineering (WISE)*, pages 64–73, 2000.

[8] M. Yoshikawa, T. Amagasa, T. Shimura, S. Uemura: Xrel: A path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology*, 1(1), June 2001.


1


2


3

1 Dr. Sven Helmer studierte Informatik an der Universität Karlsruhe und promovierte an der Universität Mannheim im Bereich Datenbanksysteme. Er ist zur Zeit als wissenschaftlicher Assistent am Lehrstuhl für Praktische Informatik III beschäftigt.
Adresse: Fakultät für Mathematik und Informatik, Universität Mannheim, D-68131 Mannheim. E-Mail: helmer@informatik.uni-mannheim.de

2 Dipl.-Inform. C.-Ch. Kanne studierte Informatik an der RWTH Aachen (1992–1998) und ist seit 1998 Mitarbeiter am Lehrstuhl für Praktische Informatik III der Universität Mannheim. Seit 2000 ist er Geschäftsführer der data ex machina GmbH, die das XBMS Natix weiterentwickelt und vermarktet.
Adresse: Fakultät für Mathematik und Informatik, Universität Mannheim, D-68131 Mannheim. E-Mail: cc@informatik.uni-mannheim.de

3 Prof. Dr. habil. Guido Moerkotte ist Inhaber des Lehrstuhls für Praktische Informatik III der Universität Mannheim und Mitgründer der data ex machina GmbH und der Xyleme SA.
Adresse: Fakultät für Mathematik und Informatik, Universität Mannheim, D-68131 Mannheim. E-Mail: moerkotte@informatik.uni-mannheim.de