FACHHOCHSCHULE WEDEL

BETRIEBSWIRTSCHAFTLICHE SEMINARARBEIT

im Fachbereich

WIRTSCHAFTSINFORMATIK

Thema:

Integration von XML in Anwendungen

Erstellt von:	Matthias Ströh Graf-Johann-Weg 15a 22459 Hamburg Tel. 040 – 60 08 55 61 Mob. 0170 – 141 79 93 Matr.-Nr. 5545
Verwaltungssemester:	7
Abgabetermin am:	21.12.2005
Referent:	Herr Prof. Dr. Sebastian Iwanowski

Inhaltsverzeichnis

Darstellungsverzeichnis

1. Einleitung

Die zunehmende Vernetzung von Computern ermöglicht es jederzeit, an jedem Ort auf elektronisch gespeicherte Informationen zuzugreifen. Da neue Informationsquellen jederzeit hinzugefügt werden können, führt dieses zu einer großen und kontinuierlich wachsenden Menge an Informationen. Für den Anwender stellt sich die Frage, wie er effizient die für ihn relevanten Informationen findet und wie er sie sinnvoll in seine Arbeitsprozesse integrieren kann. Somit ist es ein Ziel, Anwendern mit geeigneten Informationsumgebungen

- einen nahtlosen und integrierten Zugang zu heterogenen Informationsquellen zu ermöglichen
- eine effektive Nutzung von verteilen Informationsobjekten zu unterstützen
- und das aktuelle Arbeitsumfeld des Nutzer zu berücksichtigen.

Die wachsende Heterogenität von Softwareprodukten in Unternehmen fordert eine Steigerung der Interoperabilität von Anwendungen im Unternehmen und auch nach einem unkomplizierten Datenaustausch zwischen sich und anderen Unternehmen.

Um eine profitable Integration von XML in eine Anwendung zu erzielen ist es wichtig sich genügend Zeit zu nehmen und Vorausschauend zu planen und zu entwickeln. Dazu gehört viel Erfahrung und Know-how, da die Integration von XML nicht nur aus dem Hinzufügen von Tags besteht.

Die Integration von XML in eine Anwendung nimmt unter anderem Einfluss auf die Punkte

- Performanz
- Sicherheit
- Erweiterbarkeit
- Wiederverwendbarkeit
- Datenzugriff

welche im Folgenden näher betrachtet werden.

2. Wo kommt XML in einer Anwendung zum Einsatz?

In Anwendungen kann XML zum Austausch von Daten zwischen den Komponenten einer Anwendung verwendet werden. Auf welche Art und Weise der Datenfluss in einer Anwendung mit Hilfe von XML geschieht ist entscheidend für die spätere Interoperabilität mit anderen Anwendungen. In der Abbildung 1 wird Beispielhaft der Aufbau einer Anwendung mit Datenflüssen dargestellt.

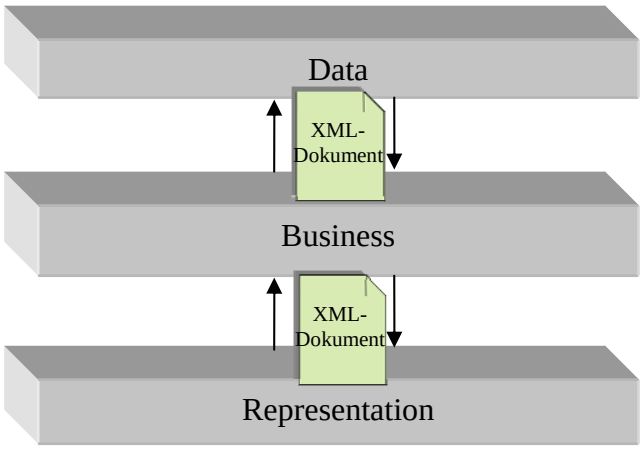

Abbildung 1: Datenstrom in einer Anwendung

Wenn XML bereits in einer Anwendung eingesetzt wird, ist eine Grundvoraussetzung für die Interoperabilität einer Anwendung geschaffen. Mittels XML-Dokumenten können dann Daten zwischen mehreren Anwendungen und zwischen den Komponenten der Anwendung ausgetauscht werden. In der Abbildung 2 wird Beispielhaft veranschaulicht, wo Datenflüsse zwischen Anwendungen und den Komponenten der Anwendung auftreten.

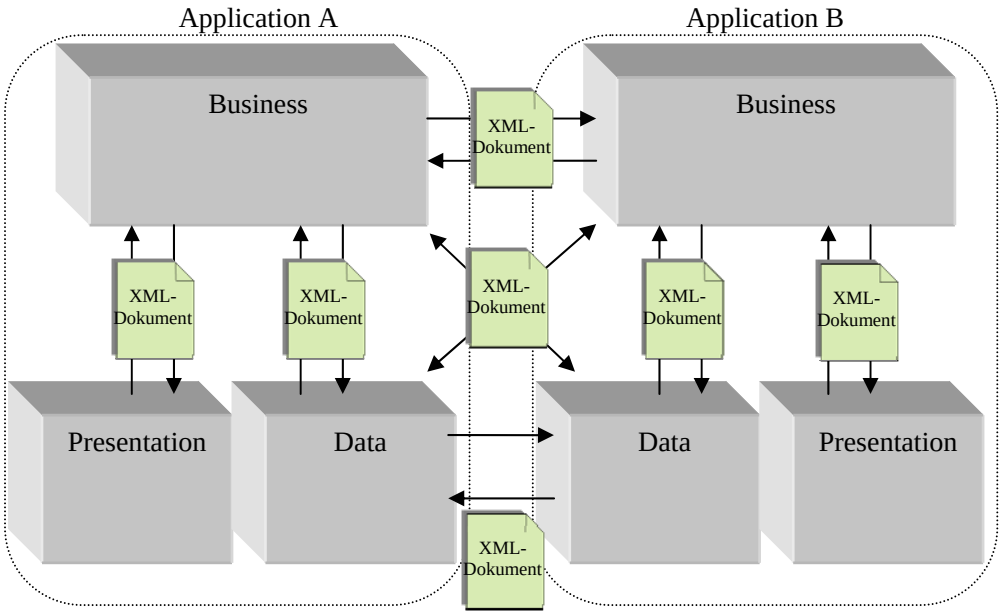

Abbildung 2: Datenstrom in einer Anwendung und zwischen einer anderen Anwendung

Weiterhin wird XML eingesetzt um Daten aus mehreren Datenquellen zusammen zu führen und somit eine gemeinsame Repräsentation zu ermöglichen. Besonderen Nutzen kann dann daraus entstehen, wenn die Datenquellen ein ungleiches Format aufweisen. Zum Beispiel können Kundendaten, Bestellungen und Rechnung in unterschiedlichen Formaten in einem Unternehmen getrennt gespeichert werden. Mit Hilfe von Konvertern werden die einzelnen Daten in XML konvertiert und zusammengeführt, wie im Abbildung 3 zusehen ist.

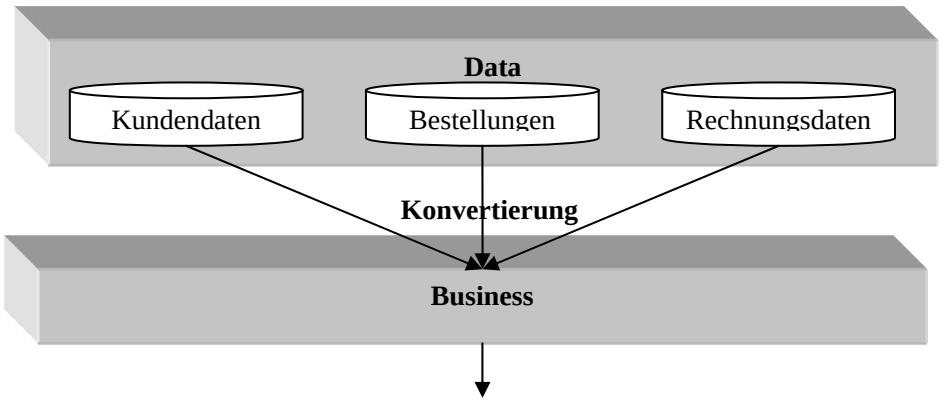

Abbildung 3: Zusammenführen mehrerer Datenquellen

3. Aufbau von Strukturen eines Schemas

Ein XML-Schema zu erstellen ähnelt dem Erstellen eines objektorientierten Hierarchiemodells oder dem Entwerfen eines Datenbankmodells. Es wird versucht die reale Welt mit den Mitteln von XML darzustellen. Hierbei ist eine repräsentative und erweiterbare Darstellung der Daten zu wählen um später Arbeit bei der Wartung und Weiterentwicklung zu sparen. Die Daten werden in Tags dargestellt. Wie granular die Tags aufsplittet werden müssen ist von einer Anwendung abhängig, die diese Daten verwendet. Umso weniger granular die Daten sind, desto mehr muss evt. eine Anwendung sich die Daten selber raussuchen. Das bedeutet zusätzliche Integration von "Mini-Parsern" in die Anwendung. Zum Beispiel könnte eine Transaktion wie Folgt repräsentiert werden: <Transaktion>Abbuchung 25 11 2005 200.00</Transaktion>
Die Anwendung müsste also wissen, dass der erste Teil der Transaktion die Art der Transaktion, der zweite Teil den Tag, der dritte Teil der Monat, der vierte Teil das Jahr und der fünfte Teil den Betrag darstellen. Die meisten Anwendungen müssten also die Daten an den Whitespaces trennen um sie weiter zu verarbeiten. Eine leichtere Verarbeitung wäre gegeben wenn der Tag wie Folgt aufgebrochen wird:

```
<Transaktion type="Abbuchung">
  <Datum>25 11 2005</Datum>
  <Betrag>200.00</Betrag>
</Transaktion>
```
[1]

Nun sind die Daten generell für eine Anwendung leichter zu verarbeiten, es bleibt nun noch die Frage, ob das Datum weiter aufgeteilt werden sollte. Dies lässt sich nicht sagen, da die Granularität von der Funktionalität der Anwendung bzw. vom Verwendungszweck der Daten abhängt. Im Allgemeinen sollte jedoch versucht werden Daten möglichst granular darzustellen, da nicht auszuschließen ist, dass Andere Anwendungen diese Daten ebenfalls verwenden.

[1] Harold, Elliotte Rusty (2004), Effectiv XML - 50 Specific Ways to Improve Your XML, S. 59 - 63

Die Werte von Attributen sollten möglichst nur unstrukturierte Daten enthalten. So wäre z.B. <polygon punkte="350,75 379,161 469,161" />
durch Unterelemente mit Attributen darzustellen. Ein solch schlechtes Design resultiert meistens aus dem Versuch die XML-Dokumente klein zu halten. Elemente haben außerdem den entscheidenden Vorteil, dass sie später leichter Erweitert werden können.

```
<polygon>
  <point x="350" y="75" />
  <point x="379" y="161" />
  <point x="469" y="161" />
</polygon>
```

Um die Flexibilität der Struktur eines Schemas zu erhalten ist es hilfreich für Tags keine Namen von Firmen, Produkten etc. zu verwenden, da sich diese mit der Zeit ändern können. Es sollte auch versucht werden große Schemata zu vermeiden. Besser sind kleine Schemata die gut wieder verwendet werden können. Rekursionen sollten in Schemata möglichst sparsam verwendet werden, da die Kontrolle über die Schachtelungstiefe verloren geht. Es sollte nach alternativen gesucht werden das Schema erweiterbar zu gestallten bevor Rekursionen zugelassen werden.

4. Datensparsamkeit zum Performanzgewinn

XML-Dokumente werden mit großer Wahrscheinlichkeit oft zwischen Anwendungen bzw. zwischen den Schichten einer Anwendung Transferiert. Die Art, wie die Daten strukturiert sind ist entscheidend für die Effizienz des Datenflusses zwischen Schichten der Anwendung bzw. zwischen den Anwendungen. Es kommt oft vor, dass eine Anwendung mehr Daten verarbeitet bzw. weitergibt, als tatsächlich benötigt. Das erzeugt überall einen unnötigen Overhead, wo das Dokument verwendet wird. Wenn die Anwendung also ein XML-Dokument bekommt, dann sollte sie zuerst die unnötigen von den nötigen Daten trennen.

Eine Methode dies zu erreichen ist der Einsatz von einem XSLT-Style-Sheet.

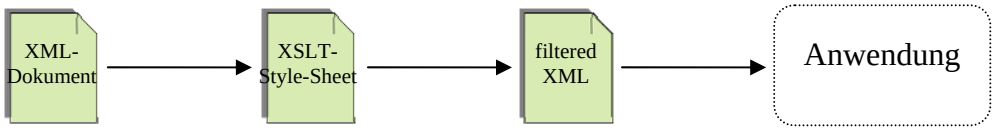

Abbildung 4: Datenfilter mit XSLT

Das spart anschließend Bandbreite, Speicher und Rechenzeit beim Parsen.
Am besten ist es jedoch diesem Problem ganz am Anfang zu begegnen und die Dokumente mit großer Sorgfalt zu entwickeln.

5. Lesbarkeit und Performanz

Durch die Flexibilität von XML lassen sich gut selbstbeschreibende Datenstrukturen erstellen. Das erhöht die Wartbarkeit und verringert Fehlinterpretationen von Personen, die mit diesen Dokumenten arbeiten. Die Flexibilität und somit die Möglichkeit leicht lesbare XML-Dokumente zu erstellen hat aber auch Schattenseiten. Z.B. würden lange Element-Typ-Namen die Größe von umfangreichen XML-Dokumenten erheblich aufblähen und sehr zum Nachteil für eine Technologie, die für Interoperabilität sorgt. Es gibt verschiedene Möglichkeiten diesem Problem zu begegnen. So könnte eine Legende als Kommentar in das XML-Dokument eingefügt werden, welches eine verkürzte Schreibweise der Element-Typ-Namen beschreibt. Es gibt aber auch Möglichkeiten Zeichen zu sparen, indem Elemente mit Unterelementen versehen werden, wie im folgenden Beispiel zusehen ist:

```
<InventarNummer>1</InventarNummer>
<InventarNummer>2</InventarNummer>      } 102 Zeichen
<InventarNummer>3</InventarNummer>

Wird zu:
<Inventar>
<Nr>1</Nr>
<Nr>2</Nr>                              } 51 Zeichen
<Nr>3</Nr>
</Inventar>
```

Abbildung 5: Beispiel für Länge von XML-Dokumenten

Eine weitere Möglichkeit die Größe von XML-Dokumenten zu reduzieren ist das Komprimieren. Komprimieren nimmt jedoch immer eine gewisse Laufzeit für sich in Anspruch, so dass das Komprimieren nicht immer sinnvoll ist. Es können jedoch auch erstaunliche Speicherersparnisse erreicht werden. So schreibt Elliotte Rusty Harold, das ein 70-Seitiges Dokument mit Screenshots und Diagrammen, welches er in Word erstellt hat, mit Hilfe von OpenOffice geöffnet und dem dort integrierten XML Komprimierungsverfahren von 6,7MB auf 522K reduziert speichern konnte.[2]

Sinnvoll kann es auch sein, zwei Versionen des XML-Dokuments zu erstellen. Eine die gut lesbar für einen Entwickler ist und die Andere, die gut für maschinelle Verarbeitung ist. Somit kann nach dem Erstellen der beiden Dokumente entschieden werden, welche der beiden Versionen der XML-Dokumente verwendet wird.

Auch wenn die Performanz wichtig ist, sollte trotzdem nicht vergessen werden ein gut lesbares XML-Dokument zu erzeugen, da dies XML unter anderem ausmacht. Das heißt es wird die richtige Mischung aus Lesbarkeit und Performanz gesucht.

6. Standardisierte Schnittstellen für Anwendungen

6.1 SAX

Die Simple API for XML, kurz SAX, ist ein Standard, der ein API zum parsen von XML-Daten beschreibt. Die aktuelle Hauptversion SAX 2.0 wurde 2000 von David Megginson veröffentlicht und ist Public Domain. Ein SAX-Parser liest XML-Daten als sequentiellen Datenstrom und ruft für im Standard definierte Ereignisse vorgegebene Callback-Funktionen auf. Eine Anwendung, die SAX nutzt, kann eigene Unterprogramme als Callback-Funktionen registrieren und auf diese Weise die XML-Daten auswerten.[3] Eine SAX-Anwendung besteht im groben aus einer Parser-Factory, verschiedenen Event-Handlern und dem eigentlichen Parser, wie in Abbildung 6 dargestellt.

[2] Harold, Elliotte Rusty (2004), Effectiv XML - 50 Specific Ways to Improve Your XML, S. 284

[3] http://de.wikipedia.org/wiki/Simple_API_for_XML, Abruf: 20.12.2005

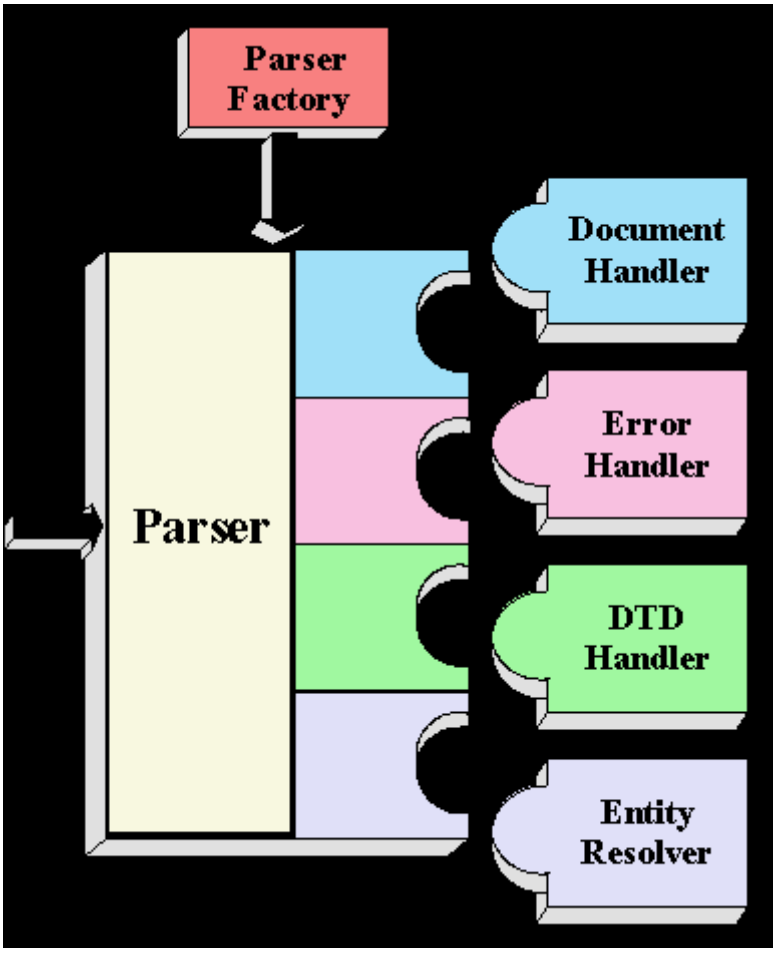

Abbildung 6: Schematischer Aufbau von SAX[4]

Ein kurzes Beispielprogramm das mit Hilfe von SAX Filmkritiken einlesen und den Durchschnittswert errechnen kann. [5]

```
<! -- Eine vereinfachte DTD für Filmkritiken -->

<!ELEMENT reviews      (movie)+>
<!ELEMENT movie        (name,review+)>
<!ELEMENT name         (#PCDATA)>
<!ELEMENT review       (#PCDATA)>
<!ATTLIST   review
            rating     CDATA #REQUIRED>
```

Abbildung 7: Beispielschema zu SAX

[4] http://www.fh-wedel.de/~si/seminare/ws00/Ausarbeitung/7.xmlparser/kap_2.html, Abruf: 20.12.2005
[5] Arciniegas, Fabio (2001), XML Developer's Guide, S. 93-94

Ein Beispieldokument:

```
<?xml version=“1.0“ ?>
        <!DOCTYPE reviews SYSTEM “reviews.dtd“>
<reviews>
        <movie>
                <name>Forest Gump</name>
                <review rating=”5”>guter Film...</review>
                <review rating=”5”>... ... ...</review>
                <review rating=”4.6”>... ... ...</review>
                <review rating=”4”>... ... ...</review>
        </movie>
</reviews>
```

Abbildung 8: Beispieldokument zu SAX

Um SAX in eine Anwendung zu integrieren sind folgende Schritte notwendig: Es müssen Handler definiert werden. Unsere Beispielanwendung muss zwei Dinge tun. Sie muss nach Daten suchen, die als Attributwert in einem Element gespeichert sind und sie muss auf das Ende eines Elements achten um dann das Ergebnis auszugeben.

```
import org.xml.sax.Parser;
import org.xml.sax.HandlerBase;
import org.xml.sax.AttributeList;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
// ********************************* Klassendefinition
public class SAXMoviesHandler extends HandlerBase
{
// ********************************* Spezielle SAX-Methoden

        public void startElement(String name, AttributList atts) {
                if(name.compareTo("review") == 0)
                        average = (numberOfReviews*average +
                        new Integer(atts.getValue(0)).intValue())+
                        numberOfReviews;
        }

        public void endElement(String name) {
                if(name.compareTo("") == 0)
                        System.out.println("The average rating for"+
                        " this movie is " + average);
        }
        private floate average;
        private int numberOfReviews;
```

Abbildung 9: Beispiel Java-Programmausschnitt zu SAX

Anschließend muss der Handler beim Parser registriert werden. Dazu wird nun eine Instanz eines SAX-Parsers mittels Parser-Factory erzeugt und der Handler, wie unten zu sehen, registriert. Zuletzt muss noch zum Starten des parsens die Methode parse aufgerufen werden, die als Parameter das zu verarbeitende XML-Dokument erwartet.

```
SAXCountHandlers handler = new SAXCountHandlers();
try {
        parser.setContentHandler(handler);
        parser.parse(xmlFile);
}
catch (SAXException e) {
        System.out.println("Error parsing " + xmlFile);
```

Abbildung 10: Beispiel Java-Programmausschnitt zu SAX (2)

6.2 DOM

Die W3C-Spezifikation des Document Object Model*s* (abgekürzt als: DOM) definiert eine Programmiersprachen-unabhängig formulierte Menge abstrakter Schnittstellen zum lesenden und schreibenden Zugriff auf gültige HTML und wohlgeformte XML-Dokumente sowie eine Reihe weiterer Formate.[6] Es stellt die Struktur eines XML-Dokuments in einer Baumstruktur dar, welche sich dann über das DOM-API auslesen oder manipulieren lässt. An dieser Stelle wird nur ein kurzes Beispiel gezeigt, da eine ausführlichere Beschreibung wie bei SAX den Rahmen dieser Ausarbeitung sprengen würde.

Beispieldokument:

```
<table>
   <thead>
     <tr>
       <th>Vorname</th>
       <th>Name</th>
     </tr>
   </thead>
   <tbody>
     <tr>
       <td>Donald</td>
       <td>Duck</td>
     </tr>
   </tbody>
</table>
```

Abbildung 11: Beispieldokument für ein DOM-Baum

[6] http://www.jeckle.de/vorlesung/xml/script.html#DOM, Abruf: 20.12.2005

Baumrepräsentation des Beispieldokumentes von DOM:

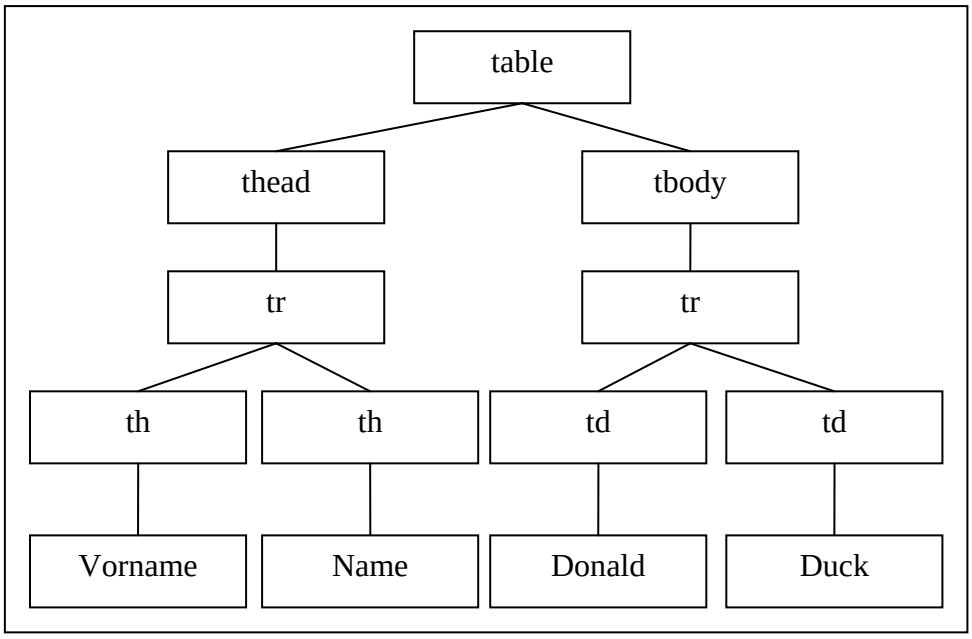

Abbildung 12: Beispiel des DOM-Baumes aus dem Bsp. Dokument[7]

Bemängelt wird bei DOM häufig, dass das komplette Dokument in den Speicher eingelesen werden muss um mit ihm zu Arbeiten. Das erlaubt zwar einen schnellen Zugriff auf das Komplette Dokument, verursacht aber bei großen Dokumenten Speicherprobleme und Performanz einbußen.

6.3 Data Binding

Data Binding ist ein neuartiger Ansatz zur Integration von XML in Anwendungen. Es werden XML-basierte Datenbeschreibungen in Objektstrukturen von Anwendungen dargestellt. Anders als SAX oder DOM stellt der Data Binding Ansatz nicht nur eine Menge an Schnittstellen zur Verfügung, sondern leitet aus einer XML-Grammatik vokabularspezifische Speicher- und Verarbeitungsstrukturen ab.[8] Somit ist Data Binding die Möglichkeit schnell und mit wenig Codeerstellung XML in eine Anwendung zu integrieren. Entwickler die Java beherrschen können schnell in ihrer

[7] http://de.wikipedia.org/wiki/Document_Object_Model, Abruf: 20.12.2005
[8] http://www.jeckle.de/vorlesung/xml/script.html#XMLDataBinding, Abruf: 20.12.2005

vertrauten Welt mit XML arbeiten. Über die in Abbildung 13 abgebildeten Schritte gelangt ein XML-Schema in die Form einer Java-Class-Datei.

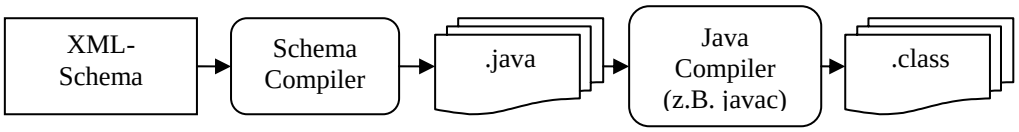

Abbildung 13: Schema zur Generation einer .class Datei mit Data Binding

6.4 Überblick

Im Folgenden wird ein kurzer Überblick über die Pros und Kontras zum Einsatz der verschiedenen vorgestellten Möglichkeiten gegeben.

	DOM	SAX	Data Binding
Nutzen	XML-Dokument mit wenig Elementen	Dokumente mit einer begrenzten Schachtelungstiefe	statische XML-Dokumente
	Änderung der Dokumentstruktur zur Laufzeit	Wenn DOM zu langsam ist	Klassenorientierte Schnittstelle erwünscht
	Zugriff auf das komplette Dokument	umfangreiche XML-Dokumente	Vereinfachung des Datenzugriffs der Programmlogik
Meiden	umfangreiche XML-Dokumente	Zugriff auf das komplette Dokument benötigt.	
		XML-Dokument mit wenig Elementen	

Abbildung 14:Tabelle über Pro und Kontras zum Einsatz von SAX, DOM oder Data Binding

7. XML-Dokumente und ihre Sicherheit

XML formatierte Daten können verschiedener Natur sein. Sie können einzig in der Firma Verwendung finden oder auch durch die ganze Welt verschickt werden. Wenn diese Daten sensible Daten sind oder deren Veränderung der Firma schaden kann, ist es gut zu wissen auf welchen Teilen der Strecke die Daten angreifbar sind.

1. Eine Gefahr geht von öffentlichen DTDs aus, die bösartiger Weise geändert wurden. Dies könnte Anwendungen auf der ganzen Welt betreffen. Es ist also ratsam kritischen Anwendungen keinen Zugriff auf öffentliche DTDs zu gestatten, die man nicht selber unter Kontrolle hat. Wenn der Server, der die öffentliche DTD zur Verfügung stelle,

keinen hohen Sicherheitsgrad aufweist, dann besteht die Möglichkeit eine lokale Kopie dieser anzufertigen.

2. Viele Firewalls sind nicht in der Lage XML-Dokumente auf ihre Integrität zu testen. Die XML-Dokumente werden meist ignoriert und durchgelassen. Das kann schlimme Folgen für das System haben, wenn XML-Daten verändert oder ersetzt werden, die dann zu Fehlern bzw. Problemen führen. Wichtig ist es deshalb, Validationsroutinen in die Anwendung zu implementieren, die dann den Inhalt jedes Elementes genau analysieren.
3. Parser und XSLT-Prozessoren können zum Teil Kontrollen zur Sicherheit in ihren Einsatzbereichen übernehmen.
4. Ungeachtet, dass XML-Dokumente eine Element- und Attributlänge benötigen sollten die Längen der Werte geprüft werden. Mit überladenen Elementen oder Attributen können Hacker den Parser, sowie Komponenten der Anwendung zum Absturz bringen.
5. Fehler, die vom Parser oder XSLT-Prozessor generiert werden, sollten eine komplette Fehlerbehandlung nach sich ziehen, die auf jegliche Art Fehler angemessen reagiert.

8. Einsatz von XML-Tools

XML kann auf viele weisen und verschieden stark von Programmen unterstützt werden. Wie gewinnbringend ein Tool eine Anwendung unterstützt muss vorher geprüft werden. Rauszufinden welches Werkzeug zu einer Anwendung passt ist oft nicht so einfach. Meistens gibt es keine Informationen darüber wie solche Tools im Detail arbeiten. Somit gilt es selbst auszuprobieren, wie solche Tools reagieren.

Dinge auf die bei Tools geachtet werden sollte:

- Bewertung und Anpassung des automatisch generierten Codes: Oft wird der Code von XML-Generatoren versteckt. Es bleibt also oft nicht die Möglichkeit die Generatoren an die Anforderungen der Anwendung anzupassen und somit die Optimale Integration von XML für diese Anwendung zu erzielen.

- Viele Anbieter solcher Generatoren fügen leider eigene Ergänzungen an die Generierten Dokumente an. Der Grund dafür ist, dass die Anbieter nicht wollen, dass der Output des Generators mit anderen Werkzeugen benutzt werden soll. Das führt nicht nur dazu, dass eine Abhängigkeit zu diesem Werkzeug entsteht, die Dokumente werden auch unnötiger weise aufgebläht.
- Eigene Datentypen beachten: Viele Produkte generieren neue Dateien für unterschiedliche Typen von Informationen um eine bestimmte Aufgabe zu erfüllen. Einige Produkte können die Abhängigkeit an den Hersteller dadurch stark erzwingen. Außerdem kann das dazu führen, dass die Plattformunabhängigkeit verloren geht.
- Schemakonversionen und Generation sind Standardfunktionen in vielen XML-Tools. Diese Tools können von selbst XML-Schemata erstellen oder von einem Schemaformat in ein anderes überführen. Diese Konversion oder Generierung sollte jedoch jedes Mal auf Korrektheit überprüft werden, da sie oft Fehler enthalten.
- Beim Verwalten von großen Dokumenten ist das Arbeiten mit einer Volltextsuche oder einer elementen- oder attributbasierten Anfrage, zum aufspüren von Teilen des Dokumentes sehr wichtig. Es gibt dafür auch die bequeme Möglichkeit mit XPath oder XQuery zu arbeiten.
- Einige Tools besitzen umfassende Schnittstellen um die Anwendung die mit XML arbeitet zu administrieren. Solche Tools sparen viel Programmieraufwand.
- Ein wichtiger, oftmals außer acht gelassener Aspekt ist die Rückmeldung eines XML-Tools. Gleicht man eine DTD und ein XML-Dokument ab, dann möchte man gerne wissen ob etwas falsch an diesen zusammengehörigen Dokumenten ist und wenn ja, auch gerne eine ausführliche Fehlerbeschreibung haben.
- Man sollte auf schon im Unternehmen vorhandene XML-Tools achten, da unter Umständen Lizenzkosten gespart werden könnten und eine reibungslose Zusammenarbeit von bereits verwendeten Tools eher gewährleistet werden kann.

9. Verwendung von DTDs oder XSDs

Die Schema Definition Language XSD ist eine neuere Schemadefinition die großen Anklang findet. Die XSD-Schema unterstützen eine Vielzahl an Funktionen, die es ermöglichen komplexe XML-Dokumente zu erstellen und zu validieren. Im Vergleich zu XSDs bieten DTDs weniger Funktionalität, was aber nicht heißt, dass DTDs obsolet sind.

9.1 Was ist ein XSD?

XML-Schema ist eine Empfehlung des W3C zum Definieren von XML-Dokumentstrukturen. Anders als bei den klassischen XML-DTDs wird die Struktur in Form eines XML-Dokuments beschrieben. Darüber hinaus wird eine große Anzahl von Datentypen unterstützt.[9]

9.2 Was ist eine DTD?

DTDs, eigentlich erstellt für SGML in Verwendung mit HTML und XML, stellen einen traditionellen Ansatz für XML-Datenvalidation dar. Dokumenttypdefinitionen sind das Mittel, mit dem wir die Struktur einer Klasse von XML-Dokumenten einschränken können.[10]

9.3 Vor- und Nachteile von DTDs und XSDs

DTDs eignen sich für XML-Dokumente deren Struktur einfach ist und bei denen sicher ist, dass die Struktur über die Zeit auch einfach bleibt. Außerdem neigen DTDs dazu kleiner zu sein als XSD-Schemata. Das bringt Vorteile, wenn der Validationscode verschickt werden soll. DTDs können also unter Umständen eine bandbreitenfreundliche Alternative zu XSDs sein. Ein weiterer Einsatzgrund von DTDs ist, dass zu integrierende XML-Tools kein XSDs unterstützen und zwangsweise DTDs mit diesem Tool verwendet werden müssen. Da DTDs um einiges älter sind, haben viele Entwickler aber auch mehr Erfahrung mit dem Umgang von DTDs als mit XSDs. Außerdem sind auf dem Markt momentan noch mehr fertige DTDs zum Erwerb und zur Wiederverwendung erhältlich als XSDs. Dieser Zustand befindet sich jedoch

[9] http://de.wikipedia.org/wiki/XSD
[10] Arciniegas, Fabio (2001), XML Developer's Guide, S. 49

im Umbruch. Ein Grund auf DTDs zurück zugreifen ist, dass XSD-Schemata mehr Rechenleistung benötigen, da sie auf Grund ihrer Komplexität mehr Overhead erzeugen. XSDs enthalten z.B. Verknüpfungen, die aufgelöst werden müssen und somit mehr Laufzeit benötigen. Eine DTD ist selbst kein XML Dokument und kann daher nicht so einfach automatisiert mit XML Tools verarbeitet werden. Bei DTDs nur angegeben werden, dass ein Element Zeichen oder bestimmte Elemente enthält, der Inhalt lässt sich aber nicht weiter spezifizieren. Wird eine Zahl verlangt, kann stattdessen auch eine Folge von Buchstaben angegeben werden.

Besser geeignet als DTDs sind XSDs, wenn eine Software von Grund auf neu entwickelt wird, da sich XSDs voraussichtlich in der Zukunft durchsetzen werden. Einen weiteren Vorteil bringen XSDs, wenn die XML-Dokumente komplex sind, da XSD sehr flexible Validationsregeln bieten. Somit erleichtern XSD-Schemata das Realisieren von komplizierten Datenrepräsentationen. Es können z.B. auch Vorteile aus unterschiedlichen Datentypen gezogen werden, die beim Arbeiten mit DTDs gänzlich fehlen. XSD-Schemata sind des Weiteren besser für eine enge Zusammenarbeit mit relationalen Datenbanken geeignet als DTDs. Bei der Arbeit mit SOAP, dem Simple Object Access Protokoll, dass das primäre Nachrichtenprotokoll in Verbindung mit Web Services darstellt, sollte der Einsatz von XSD-Schemata berücksichtigt werden, da nur XSD-Schemata von Haus aus von SOAP unterstützt werden.
Im Hinblick auf die Erweiterbarkeit haben XSD viel mehr Möglichkeiten als DTDs. Negativ bei der Arbeit mit XSDs fällt folgendes auf:

- hohe Komplexität der XSD Schemata
- geschwätzige Syntax
- die Ergebnis-Schemata sind sehr lang
- Nicht gut zum Versenden geeignet
- Können sehr laufzeitintensiv sein

9.4 Nutzung von XSDs und DTDs nebeneinander

Es gibt die Möglichkeit mit XSDs und DTDs in einem Mix gewinnbringend zu arbeiten. Hierbei könnten XSDs die Hauptrolle übernehmen und DTDs eine untergeordnete Rolle, wie z.B. die Übertragung von Daten.. Eine Anwendung, die bereits mit DTDs arbeitet, könnte mit XSDs erweitert werden.

9.5 Gemeinsame Nutzung von XSDs und DTDs

DTDs und XSD-Schemata können verwendet werden um ein und dasselbe XML-Dokument zu validieren. Das kann der Fall sein, wenn es bereits eine Basis von XML-Dokumenten gibt, die durch DTDs validiert werden und zusätzliche Anforderungen an die Validation gestellt werden, die nur durch XSD realisiert werden können. Das lässt die Strategie zu, die Anwendung Stück für Stück von DTDs auf XSDs umzustellen ohne hohe kosten oder hohen Zeitaufwand in kurzer Zeit bewältigen zu müssen.

Es kommt auch vor, dass beide Schemta-Definitionen nicht benutzt werden, da beide für die Realisierung der Anwendungen nicht mächtig genug sind oder speziellere Schemata-Definitionen geeigneter sind.

9.6 Alternativen zu XSDs oder DTDs

Es existieren neben den reinen XSD-Schemata eine Reihe von Erweiterungen bzw. Alternativen, die in freien Arbeitsgruppen entstanden sind oder durch Vertreiber entwickelt wurden. Diese Schemata können dazu beitragen mit den Schwachstellen von XSD fertig zu werden. Im Folgenden sind ein paar Beispielschemata erwähnt:

- Schematron ist eine Schemasprache die einen deutlich anderen Weg verfolgt als DTD, XSD und RelaxNG. Während diese Sprachen ein Regelwerk definieren, das präzise alle validen Dokumente beschreibt, geht Schematron zunächst davon aus, dass alle wohlgeformten XML-Dokumente auch valide sind. Durch die Regeln, die in Schematron

definiert werden, wird die Menge der validen Dokumente dann schrittweise eingegrenzt.[11]

- RELAX NG ist eine einfache, jedoch elegante Schemasprache für XML, basierend auf Murata Makotos RELAX und James Clarks TREX. Ein RELAX NG Schema spezifiziert Muster für die Struktur und den Inhalt eines XML Dokuments. Dabei ist ein RELAX NG Schema selbst ein XML Dokument, jedoch bietet es auch eine beliebte kompakte Nicht-XML-Syntax an.[12]
- Schema for Object Oriented XML (SOX) ist eine Schema Language um syntaktische und teilweise auch semantische Strukturen eines XML-Dokuments zu definieren. Erweiterungen zu DTDs sind z.B. eine Menge an Datentypen, Namensräume und polymorphischer Inhalt.[13]

Falls eine gewisse Funktionalität von diesen Validationressorcen benötigt wird und nicht komplett auf XSD verzichtet werden soll, kann die XML-Validation in verschiedene Zonen aufgeteilt werden. In jeder Zone kommt dann das Schema zum Einsatz, welches bei der Validation den Teil übernimmt, für den sie am besten geschaffen ist.

10. Erweiterte Validation von XSD-Schema

1. XSLT hat zwar nicht zur Hauptaufgabe Dokumente zu validieren, es hat aber einige Funktionen, die dabei nützlich sein können. Wenn Style Sheets XML Daten verarbeiten, werden bei einer missglückten Validation eine Reihe Fehler ausgegeben, die anschließend behandelt werden können.
2. Eigentlich sollte vermieden werden Funktionalität in die Anwendung auszulagern, wenn XML verwendet wird. Manchmal gibt es aber keine andere Möglichkeit dieses zutun, da z.B. erhöhte Anforderungen an die Validation gestellt werden, die XSD oder andere Schemata nicht leisten

[11] http://www.e-teaching.org/glossar/schematron, Abruf: 20.12.2005
[12] http://de.wikipedia.org/wiki/RELAX_NG, Abruf: 20.12.2005
[13] http://www.w3.org/TR/NOTE-SOX/#pgfId-462926, Abruf: 20.12.2005

können oder wenn es Probleme mit der Performanz gibt, die sich anders nicht lösen lassen.

3. Anwendungsspezifische Kommentare: Zusätzlich Kommentare können in die Schemata einfügen werden, die von den selbtgeschriebenen Routinen jedoch nicht als Kommentare behandelt werden, sondern als zusätzliche Validationsregeln. Der Vorteil ist, dass sämtliche Validationsregeln zusammen in einem Schema beschrieben sind. Für diesen Zeck gibt es extra eine zweite Kommentarart in XML. Es gibt als Kommentarart „documentation“, welche die normalen Kommentare beinhaltet und „appinfo“ welche z.B. die erweiterten Validationsregeln enthält.

11. Tricks für die Validation

Übervalidation vermeiden:

Wenn sich Daten durch eine Anwendung bewegen ist es üblich sie zu validieren, bevor sie verschickt werden. Es ist üblich für Anwendungen die Daten beim Senden oder Empfangen auf Integrität zu prüfen. Werden diese öfters zwischen den Anwendungen ausgetauscht, so werden sie jedes Mal neu validiert, egal ob die Daten verändert wurden oder nicht. Eine Lösung ist ein Element in jedes XML-Dokument aufzunehmen, mit einem Attribut z.B. "Validated“, das bei jeder Änderung auf „no“ gesetzt wird und nach jeder erfolgreichen Validierung auf „yes“. Somit bräuchten die XML-Dokumente nicht so häufig geprüft werden und der Overhead würde reduziert.

Gezielte Validation:

Einige XML-Dokumente haben statische Anteile, die sich im Zeitverlauf nicht ändern. Diese statischen Anteile könnten bei der Validation einmal geprüft werden und bei weiteren Validationen ausgelassen werden. Es gäbe also eine menge Laufzeit bei dieser Art Validation zu sparen. Am besten sind solche gezielten Validationen in Routinen aufgehoben in denen die Dokumente verändert werden. Dort können die veränderten Teile nach dem Erstellen auch sofort validiert werden.

12. Erstellen von Modularen und erweiterbaren XSD-Schemata

Die Möglichkeit ein Schema aus mehreren Schemata zusammen zu bauen ist ein wichtiger Bestandteil von XML. Eine gute Modularisierung bietet auch die Möglichkeit der einfachen Erweiterbarkeit. Ein weiterer Punkt der Modularisierung ist die Widerverwendbarkeit solcher Module in anderen Schemata. In XSD werden solche Schemata mit Hilfe von "Include" und "Import" in andere Schemata eingefügt. Zusätzlich gibt es noch die Möglichkeit durch "redefine" Elemente oder Attribute zu überschreiben.

Für Erweiterungen bietet das "any"-Element im XML Schema eine flexible Möglichkeit.

13. Strategien zum Integrieren von XML-Schema Administration

Das undurchdachte Erstellen von XML-Schemata führt in vielen Unternehmen zu Problemen. Viele Mitarbeiter wissen nicht, wie wichtig die Standardisierung von XML-Schemata ist. Somit läuft die Entwicklung von Anwendungen in einigen Unternehmen in den folgenden vier Schritten ab:

1. Eine Anwendung erstellen, die XML benutzt
2. Schemata erstellen die gebraucht werden
3. Versuchen zwei Anwendungen miteinander zu verbinden
4. Feststellen, das die erstellten XML-Schemata nicht zusammen passen

Dann bleiben Folgende Möglichkeiten:

1. Die Schemata von einer Anwendung an die Andere anpassen.
2. Einfügen einer Konversion der XML-Dokumente zwischen den beiden Anwendungen.
3. Weiterarbeiten mit dem Wissen das es auch mit den inkompatiblen Schemata läuft.

Diese Ansätze werden jedoch mit großer Wahrscheinlichkeit auch in der Zukunft wieder zu Problemen führen.

Schrittweises Vorgehen

Eine gute Vorgehensweise ist das Erstellten eines Pool aus Daten-Schemata für die Anwendungen, die in einem Unternehmen verwendet werden. Wenn alle

Anwendungen nach diesem Datenpool entwickelt werden, dann steht einer späteren Zusammenarbeit der verschiedenen Anwendungen nicht mehr viel im Wege.

Exemplarische Vorgehensweise:

1. Den Entwicklern muss klar gemacht werden, dass sie nicht einfach ihre eigenen Schemata erstellen können. Es sollte eine Person geben die zentral die XML-Schemata verwaltet und verwahrt. Somit wird eine überwachte Entwicklung von den Dokumentstrukturen und XML-Datenmodellen gewährleistet.
2. Identifizieren von automatisch generierten Schemata und Standardisierung der Werkzeuge. Ein großes Problem von nicht einheitlichen Dokumenten entsteht durch Frontends, wie z.B. XML Editoren oder Datenkonvertern die automatisch Schemata aus bestehenden XML-Dokumenten generieren. Diese Schemata sind oft nicht Fehlerfrei. Ein guter Weg den Überblick zu behalten ist es zu dokumentieren, wo automatisch Code erzeugt wird, wie dieser Code generiert wird und wo er wieder zum Einsatz kommt. Bei den Tools, die benutzt werden um die Schemata weiter aufzubereiten, sollte drauf geachtet werden, dass die Entwickler alle die gleichen Tools verwenden, um die Schemata einheitlich zu halten.
3. Standardisieren von offiziellen XML-Schemata. Es sollte ein nach Standards strebendes Projektteam ins Leben gerufen werden um Standarddesigns zu entwickeln und zu kontrollieren.
4. Wenn neue XML-Basierte Anwendungen entstehen, sollten diese vom XML-Datenverwahrer überprüft werden. Folgendes könnte dann geprüft werden:
    - vermeiden von doppelten Datenmodellen von den bereits existieren und die durch die neue Anwendung hinzukommen.
    - prüfen von neuen Schemata auf das Format der schon existierenden Schemata speziell Namens- und Strukturkonventionen.

- prüfen ob es Möglichkeiten gibt Schematamodule wieder zu verwenden.
- prüfen, ob das neue Schema Möglichkeiten der Modularisierung bietet.

5. XML-Schemata repräsentieren gemeinsame Daten. Die meisten solcher Daten sind bereits definiert, verarbeitet und von mehreren Instanzen im Unternehmen kontrolliert. Um die Daten-Schemata immer auf dem neusten Stand zuhalten, wird ein Organisationsprozess benötigt. Dazu müssen sich die Entwickler der Schemata eine Erlaubnis vom Datenverwahrer zum verändern der Daten holen.
6. Kommunizieren der Prozesse, Standards und Technologien: Um effektiv neue Prozesse und Rollen in einem Unternehmen einzuführen müssen Schulungen durchgeführt werden. Standards und Prozessbeschreibungen können über das Intranet bereitgestellt werden.
7. Versionskontrolle von den Schemata: Wie bei einem relationalen Datenmodel ist es wichtig bei Anwendungen, die von XML abhängen, dass wenn es Veränderungen oder Erweiterungen gibt diese auch in der Anwendungen berücksichtigt werden. Es sollte also unbedingt auf die Integrität geachtet werden. Das XSD-Schema hat Funktionen, mit dem Kernschemata erweitert oder überschreiben werden können. Dieses Feature kann hilfreich sein, es sollte jedoch mit einer gewissen Vorsicht vorgegangen werden. Solche Funktionen zu nutzen sollte auch in den Standards auftauchen.

14. Strategien zum integrieren von XML Transformationen

Es besteht oftmals der Wunsch die XML-Daten in andere Datenformate zu wandeln. XSLT ist das führende Werkzeug dazu auf dem Markt. XSLT in den XML-Mix mit aufzunehmen, öffnet somit eine Menge neuer Integrationsmöglichkeiten. XSLT ist die Abkürzung für XSL Transformation während XSL wiederum Extensible Stylesheet Language bedeutet. XSLT ist eine Programmiersprache zur Transformation von XML-Dokumenten. Es baut auf die logische Baumstruktur eines XML-Dokumentes auf und erlaubt die

Definition von Umwandlungsregeln. XSLT-Programme, so genannte XSLT-Stylesheets, sind dabei ebenfalls nach den Regeln des XML-Standards aufgebaut.[14]

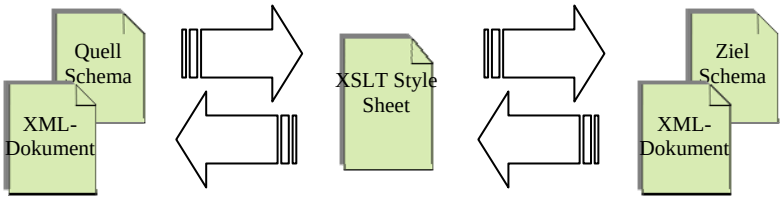

Abbildung 15: Schematische Darstellung der Funktionsweise von XSLT

14.1 Vortransformieren zum statischen Cachen

Transformieren ist sehr rechenzeitintensiv, abhängig von der Größe des zu transformierenden Dokuments und der Komplexität der Struktur. Wenn keine Zeitnahe Transformation der Daten benötigt wird, kann ein Caching-System benutzen werden um Rechenzeit zu sparen. Das heißt es werden Daten in Zeitinterwalen Transformiert und als statischen Dokumenten auf der Festplatte gespeichert oder im Hauptspeicher gehalten.

Sollte diese Möglichkeit zum Einsatz kommen, sollten folgende Punkte beachtet werden:

- bei Sensitiven Daten sollte von einem Cachen aus Sicherheitsgründen absehen werden.
- Ein Garbage-Collector sollte die veralteten Dateien wieder entfernen.

14.2 Wo liegen die Grenzen von XSLT

XSLT ist für ein breites Benutzerspektrum gemacht. Für spezielle Anwendungen mit hohen Datenaufkommen kann die Leistungsgrenze überschritten werden. Mit einem DOM-Basierter XML Parser lädt ein XSLT-Processor das komplette Dokument in den Speicher und fängt erst dann an es abzuarbeiten. Das könnte dazu führen, dass die Anwendung sehr langsam wird während die Daten vom Server geladen werden.

Einige dieser Probleme lassen sich lindern durch:

[14] http://de.wikipedia.org/wiki/Xslt, Abruf: 20.12.2005

- Einführung eines Warteschlangensystems zur Kontrolle der Transformationslast
- modellieren der XML-Dokumente in eine gut handhabbare Größe

Wenn dies nicht hilft muss evt. von XSLT abstand genommen werden und Routinen selber entwickelt werden. Alternativ kann auch ein Hardwarebeschleuniger speziell für XML angeschafft werden.

15. Möglichkeiten der Performanzsteigerungen von Anwendungen

Eine Möglichkeit ist, wie eben erwähnt einen Hardwarebeschleuniger zu installieren der speziell XML beschleunigt.

Falls Performanzprobleme auftreten, bietet es sich an seinen eigenen Parser in einer abgespeckten Version zu implementieren und die Standard-Parser zu entfernen. Dabei ist jedoch Vorsicht geboten, da alle Integritätsregeln weiterhin bestehen bleiben müssen um die Integrität zu sichern.

Strategische Redundanzen:

Ein Nachteil einer Standard-XML-Architektur ist, dass keine Indexmechanismus von Haus aus integriert ist um Datenmanipulationen zu beschleunigen. Das führt dazu, dass die Performanz von einer Anwendung, die verschiedene Sichten auf dieselben Daten benötigt, verschlechtert wird, wenn die zur Laufzeit generiert werden müssen.

Es gibt auch verschiedene Produkte, die wie eine Datenbank indizierten Zugriff und Warteschlangenfunktionen besitzen, diese sind einfacher und günstiger als sie selbst zu implementieren.

Wenn vor Aufruf einer Datensicht feststeht, dass diese häufiger von der Anwendung erstellt werden muss, kann diese auf der Festplatte als separates XML-Dokument gespeichert werden. Das fordert dann von der Anwendung, dass die Redundanten Daten automatisch aktualisiert werden, wenn Informationen in einem in Beziehung stehenden Dokument geändert werden.

Diese Vorgehensweise geht zwar gegen die bestehenden Datenmodellierungsprinzipien, in der Redundanz vermieden werden sollte, manchmal ist es jedoch sinnvoll z.B. eine Datenbank zu denormalisieren um

eine bessere Performanz zu erzielen. Bei der Arbeit sollte aber folgendes nicht außer Acht gelassen werden:

- Es sollte nicht zuviel Redundanz entstehen. Zu viele miteinander verknüpfte Dokumente würden die Laufzeit wieder verschlechtern.
- Alle Dokumente die Redundanzen enthalten sollten zu einer logische Gruppe zusammengefasst werden. Es sollen keine weiteren Redundanten Dokumente aus Dokumenten dieser Gruppe entstehen.
- Das Ziel sollt es sein die Laufzeit zu verbessern. Z.B. wenn aus mehreren Dokumenten Informationen zusammengetragen werden um sie gemeinsam darzustellen.

Literaturverzeichnis

Literaturverzeichnis

Arciniegas, Fabio (2001), XML Developer's Guide

Erl, Thomas (2004), Service-Oriented Architectur - A Field Guide to Integrating XML and Web Services

Harold, Elliotte Rusty (2004), Effectiv XML - 50 Specific Ways to Improve Your XML

Quellen im Internet

http://de.wikipedia.org/wiki/Simple_API_for_XML, Abruf: 20.12.2005

http://www.jeckle.de/vorlesung/xml/script.html#DOM, Abruf: 20.12.2005

http://de.wikipedia.org/wiki/Document_Object_Model, Abruf: 20.12.2005

http://www.jeckle.de/vorlesung/xml/script.html#XMLDataBinding, Abruf: 20.12.2005

http://www.e-teaching.org/glossar/schematron, Abruf: 20.12.2005

http://de.wikipedia.org/wiki/RELAX_NG, Abruf: 20.12.2005

http://www.w3.org/TR/NOTE-SOX/#pgfId-462926, Abruf: 20.12.2005

http://de.wikipedia.org/wiki/XSD, Abruf: 20.12.2005

http://de.wikipedia.org/wiki/Xslt, Abruf: 20.12.2005