XML-Grundlagen

Christian Weber
c_weber@informatik.uni-kl.de

Dies ist eine Einführung in die eXtensible Markup Language. Es wird der grundlegende Aufbau und die Syntax von XML-Dokumenten vorgestellt und ein Überblick über das XML-Verarbeitungsmodell und damit verbundene Kerntechnologien gegeben.

1. Einleitung - Was ist XML ?

XML (eXtensible Markup Language, [4]) ist ein vom W3C[1] eingeführter Standard zur Dokumentenauszeichnung. Nachfolgende Punkte stellen einige grundlegende Eigenschaften von XML dar:

1. **XML ist eine Methode zur Speicherung strukturierter Daten in Textform**
   Strukturierte Daten, wie sie zum Beispiel in Adressbüchern, Kalkulationsprogrammen, Vektorgrafikprogrammen oder ähnlichen Programmen anfallen, können entweder in einem binären Format oder im Textformat abgespeichert werden. XML bietet eine Menge von Regeln um derartige Daten eindeutig im Textformat zu speichern. Die Daten werden als Text in XML-Dokumente eingefügt und von Textauszeichnungen umgeben, die die Daten beschreiben. Dabei muss eine bestimmte Syntax eingehalten werden. Diese Dokumente lassen sich leicht generieren und genauso einfach maschinell verarbeiten. Dabei werden Probleme wie Erweiterbarkeit, Internationalisierung und Plattformabhängigkeit geschickt umgangen. XML eignet sich zur Speicherung und Austausch von Daten, die sich sinnvoll als Text dargestellt werden können. Zur Speicherung großer Bitfolgen, wie sie zum Beispiel bei Multimediadaten wie Bildern, Ton oder auch Videos vorkommen, ist XML ungeeignet.
2. **XML sieht aus wie HTML, ist aber kein HTML**
   Genauso wie in HTML (HyperText Markup Language, [34]) verwendet XML sogenannte Tags und Attribute. In HTML ist die Bedeutung eines jeden Tags und Attributes genau festgelegt. Dies ist bei XML nicht der Fall. Da XML eine Meta-Auszeichnungssprache ist, sind keine Tags fest vorgegeben. Der Entwickler einer Anwendung kann die Menge und Bedeutung der Tags, die er verwenden möchte, selbst definieren.
3. **XML ist Text, aber nicht zum Lesen**
   XML-Dateien sind Textdateien, die nicht dafür gedacht sind von Menschen gelesen zu werden. Es gibt jedoch auch Situationen in denen dies notwendig ist. So kann man auf sehr einfache Weise nach Fehlern in einer beschädigten XML-Datei suchen, da man die Datei in einem beliebigen Texteditor öffnen kann. Während Fehler in einer HTML-Datei meist toleriert werden, muss nach XML-Spezifikation eine Anwendung die Verarbeitung einer XML-Datei stoppen und einen Fehler ausgeben, wenn die Datei defekt ist. XML bietet ein Datenformat, das über lange Zeit lesbar bleibt.

1 World Wide Web Consortium
http://www.w3.org/

Oft war es so, dass binäre Datenformate mit einem Programm erstellt wurden, eine spätere Version des gleichen Programms diese Daten jedoch nicht mehr korrekt lesen konnte. Auf diese Art und Weise sind in den späten 60ern und frühen 70ern viele Daten von Mondlandungen verloren gegangen, da heute niemand mehr weiß in welchem binären Format die Daten auf den Magnetbändern gespeichert sind.

4. **XML ist ausführlich**
   Da XML Tags benutzt um die in Textform vorliegenden Daten von einander zu trennen, sind XML-Dateien meist wesentlich größer als binäre Formate. Dies war jedoch eine bewusste Einscheidung der XML-Entwickler, da eine Speicherung im Textformat Vorteile wie einfache Verarbeitung, manuelle Editierbarkeit und leichtere Erweiterbarkeit mit sich bringt. Die Nachteile die durch die Dateigröße entstehen, können recht einfach ausgeglichen werden. Festplattenkapazität ist heute nicht mehr so teuer wie noch vor einigen Jahren. Außerdem lassen sich die Dateien mit Komprimierungsprogrammen, die für fast alle Plattformen kostenlos erhältlich sind, sehr schnell und effektiv komprimieren. Bei der Datenübertragung können Kommunikationsprotokolle die Dateien automatisch komprimieren, so dass sie nicht mehr Bandbreite verbrauchen als ein binäres Dateiformat.
5. **XML ist eine Familie von Technologien**
   Neben der XML 1.0 Spezifikation, in der Tags und Attribute definiert sind, gibt es eine ständig wachsende Menge von Standards, die auf der XML-Spezifikation aufbauen und die Arbeit mit XML-Dokumenten erleichtern. Da die neu entwickelten Technologien auf der XML-Spezifikation aufbauen, behalten ältere XML-Anwendungen ihre Gültigkeit.
6. **XML ist neu, aber nicht so neu**
   Die Entwicklung von XML begann 1996 und wurde 1998 ein W3C-Standard. Grundlage der heutigen Auszeichnungssprachen XML und HTML, die seit 1990 entwickelt wurde, ist die Auszeichnungssprache SGML (Standard Generalized Markup Language, [35]). SGML ist eine herstellerunabhängige international normierte Dokumentenbeschreibungssprache für die logische Struktur und Inhalt von Dokumenten. 1986 wurde dieser Standard zur ISO-Norm 8897. Da HTML als generelles Repository-Format nicht umfassend genug war und SGML für den breiten Einsatz zu kompliziert war, wurde von den Entwicklern von XML eine Teilmenge von SGML als einfachere Metasprache definiert. Der Spezifikationsumfang von XML beträgt ca. 1/10 des Umfangs der SGML Spezifikation. Außerdem enthält XML mehr Restriktionen, so das die Implementierung geeigneter Verarbeitungsprogramme (z.B. Parser) wesentlich einfacher ist. Abbildung 1 zeigt den ungefähren Zusammenhang zwischen SGML, XML und der Anwendung HTML.

Abbildung 1: Zusammenhang von HTML, XML und SGML

7. **XML ist modular**
   XML erlaubt es ein neues Dokument als Kombination anderer Dokumente zu definieren. Da die einzelnen Dokumente eventuell unabhängig voneinander entwickelt wurden, ist es möglich, dass diese Dokumente die gleichen Tags oder Attribute mit unterschiedlicher Bedeutung verwenden. Um derartige Missverständnisse bei der Kombination von Dokumenten zu vermeiden, gibt in XML sogenannte Namensräume (siehe Kapitel 2.5).
8. **XML ist lizenzfrei, plattformunabhängig und gut unterstützt**
   Da XML lizenzfrei ist, kann es beliebig in einer Anwendung eingesetzt werden. Inzwischen gibt es eine zahlreiche Sammlung von Werkzeugen zum Umgang mit XML Dokumenten. Diese Sammlung ist inzwischen so groß, dass man nicht auf einen bestimmten Anbieter angewiesen ist. Da die Daten im Textformat abgespeichert werden und es für alle Plattformen Werkzeuge gibt, die XML-Dokumente bearbeiten können, ist man mit XML als Datenformat plattformabhängig.

2. XML-Spezifikation

Dieser Abschnitt befasst sich mit dem Aufbau eines XML-Dokumentes und der XML-Syntax. Es wird der grundlegende Aufbau von Dokumenttyp-Definitionen erklärt, Namensräume werden eingeführt, und der Unterschied zwischen wohlgeformten und gültigen XML-Dokumenten wird erklärt.

2.1 XML-Dokumentstruktur

Ein XML-Dokument besteht aus einem optionalen Kopf und den eigentlichen Daten. Der Kopf sollte die XML-Deklaration enthalten und eventuell eine Deklaration, die den Dokumenttyp festlegt. Die Deklaration des Dokumenttyps kann entweder eine Referenz auf eine externe Dokumenttyp-Definition (DTD) sein, oder als interne DTD direkt bei den Daten stehen (siehe Kapitel 2.3).

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
```

Beispiel 1: XML Deklaration

Falls eine XML-Deklaration im Dokument vorhanden ist, so muss sie direkt am Anfang des Dokumentes stehen. Eine XML-Deklaration enthält die Attribute `version`, `encoding`, und `standalone`.

Die `version` der XML-Deklaration hat derzeit immer den Wert `1.0`. Falls es irgendwann mal eine neue Version von XML geben wird, kann man dann anhand dieses Attributes erkennen, um welche XML-Version es sich handelt. Es kann aber auch sein, dass sich die Version nie ändern wird, da XML 1.0 eine Spezifikation ist, die mit Blick auf Erweiterbarkeit entworfen wurde. Erweiterungen von XML 1.0 wurden bisher auf Schichten aufgebaut, die von XML 1.0 zur Verfügung gestellt werden. Auf diese Weise werden existierende Dokumente und Werkzeuge nicht unbrauchbar.

Das `encoding` Attribut gibt an, welche Kodierung der Zeichen im Dokument benutzt wird. Falls das Attribut nicht angeben wird, wird davon ausgegangen, dass das Dokument die UTF-8-Kodierung mit variabler Länge des Unicode-Zeichensatzes verwendet. Der im Beispiel 1 angegebene Zeichensatz "`ISO-8859-1`" enthält Zeichen, die im Unicodezeichensatz nicht vorhanden sind, wie zum Beispiel ä, ü und ö.

Das Attribut `standalone` gibt an, ob eine Applikation möglicherweise auf eine externe DTD zugreifen muss, um die korrekten Werte für bestimmte Teile des Dokumentes zu ermitteln. Wenn es sich um Dokumente handelt, die über keine DTD verfügen oder die DTD bereits im Dokument integrieren, kann

der Wert von `standalone` auf `yes` gesetzt werden. Anderenfalls muss der Wert auf `no` gesetzt werden. Da das Attribut `standalone` optional ist, wird der Wert `no` angenommen, falls das Attribut nicht vorhanden ist.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<seminar>
  <titel>XML und Datenbanken</titel>
  <semester>WS 2002/03</semester>
  <thema>
    <name>XML-Grundlagen</name>
    <vortragender>Christian Weber</vortragender>
    <datum>24.01.2002</datum>
    <quelle>http://www.w3.org/XML/</quelle>
    <quelle>http://www.w3.org/TR/1998/REC-xml-19980210</quelle>
  </thema>
  <thema>
    <name>XML-Verarbeitungsmodelle und Language Bindings</name>
    <vortragender>Christian Müller</vortragender>
    <quelle>http://www.w3.org/XML/</quelle>
  </thema>
</seminar>
```

Beispiel 2: Einfaches XML-Dokument

Beispiel 2 zeigt ein einfaches XML-Dokument mit Kopf und Daten. Wie bereits im Kapitel 1 angesprochen, werden die Daten in Textform zwischen Auszeichnungen, den so genannten Tags, gespeichert. XML-Dokumente weisen eine hierarchische Struktur auf und können in Komponenten aufgegliedert werden, die selbst wieder aus verschiedenen Komponenten bestehen können. Diese logischen Komponenten eines Dokumentes nennt man in XML Elemente. Abbildung 2 zeigt die Baumdarstellung des Beispiels 2. Da die Dokumente eine baumartige Struktur aufweisen, nennt man das Element, das alle anderen Elemente enthält (z. B. `seminar`), auch Wurzelelement. Dieses Wurzelelement kann Kinderelemente haben (z. B. `titel`, `semester`, `thema`), die selbst wiederum Kinderelemente haben können. Jedes Element mit Ausnahme des Wurzelelementes besitzt genau ein Elternelement. Man spricht auch von einer Eltern-Kind-Beziehung. Elemente dürfen sich nicht überschneiden. Dies bedeutet, das alle Elemente die innerhalb eines Elementes (z. B. `seminar`) geöffnet werden, erst wieder geschlossen werden müssen bevor das Elternelement (z. B. `seminar`) geschlossen wird.

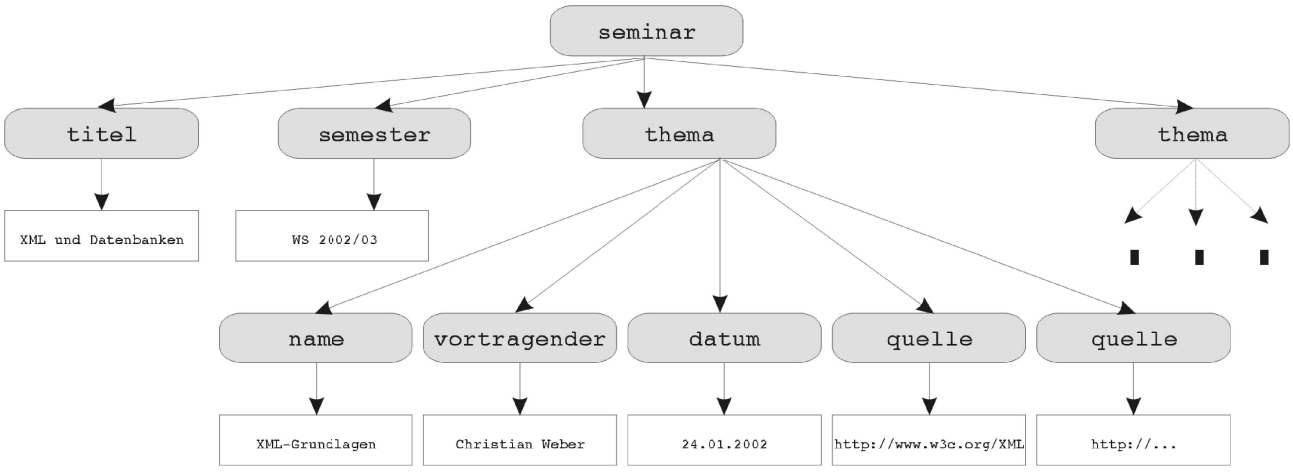

Abbildung 2: Baumstruktur des Beispiels 2

2.2 XML-Syntax

Die einzelnen Elemente eines XML-Dokuments beginnen mit einen Start-Tag (z.B. `<titel>`) und einem End-Tag (z.B. `</titel>`). Die Zeichen zwischen einem Start-Tag und End-Tag werden Inhalt genannt. Es kann auch Elemente ohne Inhalt geben (`<title />`). Im Gegensatz zu HTML wird bei XML zwischen Groß- und Kleinschreibung der Tags unterschieden. Bei der Benennung der Tags kann eine beliebige Kombination aus Groß- und Kleinbuchstaben verwendet werden, solange das Start-Tag und das End-Tag übereinstimmen. In XML dürfen Elementnamen und Attributnamen beliebige alphanumerische Zeichen, sowie die drei Interpunktionszeichen (Unterstrich, Bindestrich und Punkt) enthalten. Doppelpunkte sind auch erlaubte Zeichen, die jedoch zur Kennzeichnung von Namensräumen verwendet werden. XML-Namen müssen mit einem Buchstaben oder einem Unterstrich beginnen. Eine Längenbeschränkung für XML-Namen existiert nicht.

XML-Elementen können mit Hilfe von Attributen zusätzliche Eigenschaften zugeordnet werden. Attribute sind Tupel von Namen und Werten, die einem Start-Tag eines Elementes zugewiesen werden. Dabei wird der Name von dem Wert durch ein Gleichheitszeichen getrennt, der Wert wird in einfache oder doppelte Anführungszeichen eingeschlossen.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<seminar semester="WS 2002/03">
  <titel>XML und Datenbanken</titel>
  <thema>
    <name>XML-Grundlagen</name>
    <vortragender matnr="123456">Christian Weber</vortragender>
    <datum>24.01.2002</datum>
    <quelle>http://www.w3.org/XML/</quelle>
    <quelle>http://www.w3.org/TR/1998/REC-xml-19980210</quelle>
  </thema>
  <thema>
    <name>XML-Verarbeitungsmodelle und Language Bindings</name>
    <vortragender>Christian Müller</vortragender>
    <quelle>http://www.w3.org/XML/</quelle>
  </thema>
</seminar>
```

Beispiel 3: Einfaches XML-Dokument mit Attributen

Beispiel 3 zeigt das bereits aus Beispiel 2 bekannte Dokument, in dem das Element `semester` durch ein entsprechendes Attribut ersetzt worden ist. Beim Betrachten der beiden Beispiele stellt sich die Frage, wann man Kinderelemente oder Attribute verwendet, um Informationen darzustellen. Darüber gibt es viele Diskussionen. Einige Leute sind der Meinung, dass Attribute für Metadaten des Elementes gedacht sind. Die Gegner dieser Meinung argumentieren, dass es nicht immer einfach ist Daten und Metadaten zu unterscheiden. (siehe [2]) Eine elementbasierte Struktur ist wesentlich flexibler und leichter erweiterbar. Man kann grob festhalten, dass Elemente immer dann notwendig sind, wenn eine weitere Strukturierung notwendig ist, sie in unterschiedlichen Kontexten vorkommen oder ein wiederholtes Vorkommen notwendig ist. Attribute werden verwendet, wenn bestimmte Datentypen (z.B. ID/IDREF) benutzt werden sollen. Für bestimmte Anwendungen sind Attribute angenehmer zu verarbeiten als Elemente. Im Prinzip bleibt es dem Anwender überlassen, welche Daten er mit Attributen und welche er mit Elementen beschreibt.

In der Zeichenkette des Inhalts eines Elementes oder des Wertes eines Attributes dürfen gewisse Zeichen nicht auftreten, da sie in XML-Dokumenten eine bestimmte Bedeutung haben und sie daher beim Parsen falsch interpretiert werden würden. Diese Zeichen müssen durch Entity-Referenzen ersetzt werden. Die fünf in XML vordefinierte Entity-Referenzen sind in Tabelle 1 dargestellt.

Wenn in einem XML-Dokument Abschnitte vorkommen, in dem viele Zeichen durch Entities ersetzt werden müssen, kann man die Zeichenkette auch in einen CDATA-Abschnitt einschließen. CDATA

steht für character data. Ein CDATA-Abschnitt wird durch "<![CDATA[" und "]]>" eingeschlossen. Alles was nur zwischen "<![CDATA[" und "]]>" eingeschlossen ist wird als reine Zeichendaten und nicht als Auszeichnungen interpretiert. Das einzige, was in einem CDATA-Abschnitt nicht auftauchen darf, ist die Endmarkierung eines CDATA-Abschnitts.

Entity	
<	Kleiner Zeichen (<)
&	Ampersand (&)
>	Größerzeichen (>)
"	Doppelte Anführungszeichen (")
'	Einfache Anführungszeichen (')

Tabelle 1: vordefinierte Entities

Genau wie in HTML-Dokumenten gibt es auch bei XML die Möglichkeit Kommentare in ein Dokument zu schreiben. Die Syntax der Kommentare ist die gleiche wie in HTML-Dokumenten. Ein Kommentar beginnt mit "<!--" und endet mit "-->". Kommentare dürfen, außer innerhalb eines Tags, an jeder Stelle eines Dokumentes auftreten.
Ebenfalls können in XML-Dokumenten Processing Instructions (<? instructions ?>) vorkommen, die von einer Anwendung, die das XML-Dokument liest, verarbeitet werden können. Eine bekanntes Beispiel dafür ist die Scriptsprache PHP (PHP: Hypertext Preprocessor, [37]).

2.3 Dokumenttyp-Definitionen (DTD)

Gültige XML-Dokumente entsprechen dem XML-Standard und enthalten eine Dokumenttyp-Deklaration, die die Dokumenttyp-Definition (DTD) kennzeichnet, der das Dokument entsprechen muss. Eine DTD ist die Grammtik für die spezielle XML-Sprache, die in der Anwendung benutzt wird. Die DTD listet alle Elemente, Attribute und Entities auf, die im Dokument benutzt werden dürfen, sowie den Kontext, in dem sie auftauchen. Alles was in einer DTD nicht explizit erlaubt wird, ist verboten. Eine DTD kann entweder in einer externen Datei stehen oder selbst Bestandteil des Dokumentes sein. Nachdem die folgenden Dokumenttyp-Deklaration in das Beispiel 3 eingefügt wurde, kann das Beispiel mit der DTD aus Beispiel 5 validiert werden.

```
<! DOCTYPE seminar SYSTEM "http://wwwdbis.informatik.uni-kl.de/seminar.dtd">
```

Beispiel 4: Dokumenttyp-Deklaration

Diese Dokumenttyp-Deklaration gibt an, dass seminar das Wurzelelement des Dokumentes ist und das die DTD unter dem Uniform Resource Identifier (URI, [38]) http://wwwdbis.informatik.uni-kl.de/seminar.dtd zu finden ist. Die URI kann auch ein Pfad im lokalen Dateisystem sein.

```
<!ELEMENT seminar (titel,thema*)>
<!ATTLIST seminar semester CDATA #REQUIRED>
<!ELEMENT titel  (#PCDATA)>
<!ELEMENT thema  (name,vortragender,datum?,quelle+)>
<!ELEMENT name   (#PCDATA)>
<!ELEMENT thema  (#PCDATA)>
<!ELEMENT vortragender (#PCDATA)>
<!ATTLIST vortragender matnr CDATA #IMPLIED>
<!ELEMENT datum  (#PCDATA)>
<!ELEMENT quelle (#PCDATA)>
```

Beispiel 5: DTD für Beispiel 1

In einer DTD ist es auch möglich anzugeben wie oft ein Element an einer bestimmten Stelle vorkommen darf. Dazu wird an den Elementnamen ein bestimmtes Suffix angehängt. Es gibt drei verschiedenen Suffixe um die Häufigkeit eines Elementes anzugeben:

- ? kein oder ein Element (z.B. `datum`)
- * kein oder beliebig viele Elemente (z.B. `thema`)
- + ein oder mehrere Elemente (z.B. `quelle`)

Auch die Reihenfolge der Elemente wird durch das Inhaltsmodell (z.B. `titel, thema*`) festgelegt. Außerdem gibt es die Möglichkeit, dass nicht alle Elemente einer Menge, sondern nur eins aus der Menge vorkommen muss. Es wäre zum Beispiel denkbar, dass man von einem Vortragenden den Namen und entweder die Telefonnummer oder eine E-Mail-Adresse haben möchte. Dies könnte man durch die folgendes DTD-Fragment ausdrücken.

```
<!ELEMENT vortragender (vorname, nachname,(telefon | email))>
<!ELEMENT vorname      (#PCDATA)>
<!ELEMENT nachname     (#PCDATA)>
<!ELEMENT telefon      (#PCDATA)>
<!ELEMENT email        (#PCDATA)>
```

Beispiel 6: DTD mit Auswahlmöglichkeit

Mit Hilfe der DTD kann man neben den fünf in XML bereits vorhanden Entity-Referenzen selbst Entities definieren. Dabei gibt es die Möglichkeit ein Entity intern oder extern zu definieren. Durch die Benutzung von Entities können Dokumente besser strukturiert und geändert werden. Teile, die an mehreren Stellen verwendet werden, können zum Beispiel in eigene Dateien ausgelagert werden. Daten, die häufig in einem Dokument an verschieden Stellen verwendet werden und auch häufig geändert werden müssen, können durch Entities ersetzt werden, um die Änderungen zu vereinfachen. Beim Parsen des Dokumentes wird das Entity vom XML-Prozessor durch den entsprechen Inhalt ersetzt. Beispiel 7 zeigt die Verwendung von exteren Entities.

```
<?xml version="1.0" encoding="ISO-8859-1">
  <!DOCTYPE lehrangebot [
    <!ELEMENT lerangebot (vorlesung*,seminar*)>
    <!ELEMENT vorlesung (titel,semester,zusammenfassung)>
    <!ELEMENT seminar (titel,semester)>
    <!ELEMENT titel (#PCDATA)>
    <!ELEMENT semester (#PCDATA)>
    <!ELEMENT zusammenfassung (#PCDATA)>
    <!ENTITY sem "WS 2002/03">
    <!ENTITY dbaw SYSTEM "http://wwwdbis.informatik.uni-kl.de/DBAW.txt">
    <!ENTITY wws  SYSTEM "http://wwwdbis.informatik.uni-kl.de/WWS.txt">
  ]>
<lehrangebot>
  <vorlesung>
    <titel>Datenbankanwendung</titel>
    <semester>&sem;</semester>
    <zusammenfassung>&dbaw;</zusammenfassung>
  </vorlesung>
  <vorlesung>
    <titel>Workflow und Web Services</titel>
    <semester>&sem;</semester>
    <zusammenfassung>&wws;</zusammenfassung>
  </vorlesung>
</lehrangebot>
```

Beispiel 7: Verwendung von Entities

Es gibt noch eine ganze Reihe weiterer Möglichkeiten die Struktur eines XML-Dokumentes mit Hilfe

einer DTD einzuschränken. Auf diese Möglichkeiten einzugehen wird an dieser Stelle verzichtet und auf das Seminarthema „Repräsentation von Struktur“ verwiesen.

2.4 Gültigkeit und Wohlgeformtheit von XML-Dokumenten

Wenn man die Korrektheit von XML-Dokumenten betrachtet, muss man zwei Fälle unterscheiden:
- wohlgeformte Dokumente
- gültige Dokumente

Wohlgeformte Dokumente entsprechen dem XML-Standard, d.h., sie müssen die folgenden Regeln einhalten:
- Die XML-Deklaration muss am Anfang des Dokuments stehen
- Existenz genau eines Wurzelelements
- Jedes Start-Tag besitzt auch ein End-Tag
- Elemente dürfen sich nicht überschneiden
- Attribute müssen in Anführungszeichen stehen
- Attribute eines Elementes müssen unterschiedliche Namen haben
- Bestimmte Zeichen innerhalb der Daten müssen durch Entities ersetzt werden

Diese Liste ist nur ein Teil der Regeln die beachtet werden müssen, damit es sich um ein wohlgeformtes XML-Dokument handelt. Ein Dokument ist gültig, falls es wohlgeformt ist und einer angegebenen DTD entspricht. Ein Parser, der ein XML-Dokument liest, muss zunächst prüfen, ob das Dokument wohlgeformt ist. Wird ein Fehler gefunden, so muss dieser der Anwendung mitgeteilt werden. Es ist nicht erlaubt, eigenständig Fehler zu beheben, egal wie leicht dieser Fehler zu erkennen ist. Mit dieser Maßnahme soll ein Kompatibilitätskrieg, wie er bis heute bei Webbrowsern besteht, vermieden werden.

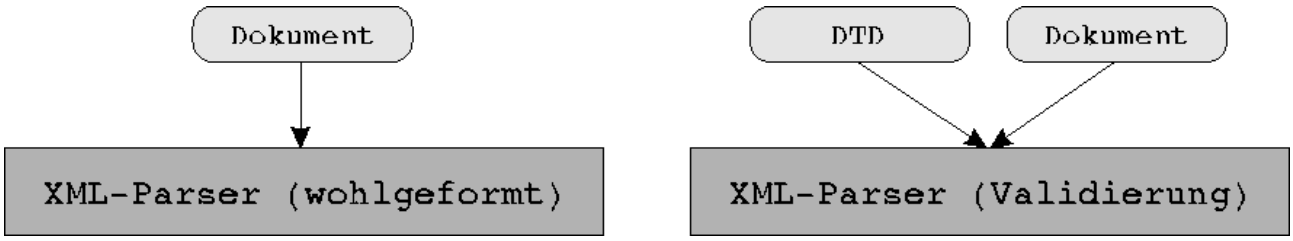

Abbildung 3: Parser

Nicht jeder Parser, der ein XML-Dokument verarbeitet, muss die Gültigkeit eines Dokumentes überprüfen. Falls der Parser die Dokumente überprüft, werden die Fehler an die Anwendung übergeben, an die der Parser auch die Daten, die im Dokument gelesen wurden, liefern soll. Es bleibt dann der Anwendung überlassen, was sie mit den Fehlern macht.

2.5 XML-Namensräume

Die Namensgebung in XML-Dokumenten bleibt völlig dem Entwickler überlassen. In manchen Dokumenten kommen Auszeichnungen aus verschiedenen Dokumenten zum Einsatz, so dass Namenskonfikte nicht auszuschließen sind. Dies kann zu Problemen mit der Interpretation und der Valdierung geben, da Elemente aus verschiedenen Dokumenten unterschiedliche Inhaltsmodelle haben können. Dieses Problem kann durch die Bindung von Elementen und Attributen an einen Uniform Resource Identifier (URI) gelöst werden. Der auf diese Art eingerichtete Namesraum ist eine Sammlung von Namen für Elementtypen und Attribute, die über einen URI eindeutig identifizierbar ist. Elemente und Attribute mit gleichen Namen aber unterschiedlicher URI sind unterschiedlich. Nur wenn sowohl der Namen und der URI übereinstimmen sind die Elemente bzw. Attribute gleich. Der verwendete URI

hat außer der Identifikation keine weitere Bedeutung. Auch wenn Beispiel 8 zu der Annahme verleitet, das unter der URL (Uniform Resource Locator) `http://wwwdbis.informatik.uni-kl.de/vorlesung` ein Dokument mit weiteren Details vorhanden ist, so muß der URI nicht zwangsläufig auf eine vorhandene URL verweisen. URL's werden nur Kennzeichnung von Namesräumen benutzt weil sie weltweit eindeutig sind.

In URI's sind oft Zeichen vorhanden, die in XML-Namen nicht zulässig sind. Aus diesem Grund werden URI's durch Präfixe ersetzt. Ein Präfix muss in dem Element definiert werden, in dem es zum ersten mal verwendet wird. Ein Präfix wird durch einen Doppelpunkt vom Element- oder Attributnamen getrennt. Ein Präfix darf aus allen Zeichen, die für XML-Namen zulässig sind, mit Ausnahme des Doppelpunktes bestehen. Präfixe die mit den Buchstaben xml beginnen, sind für die Benutzung von XML und seinen Komponenten reserviert. Die Groß- und Kleinschreibung spielt dabei keine Rolle.

Normalerweise gibt es für jede Anwendung einen Namensraum. Einige Anwendungen benutzen auch mehrere Namensräume um unterschiedliche Teile einer Anwendung besser von einander unterscheiden zu können.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes">
<lehrveranstaltungen>
  <vor:vorlesung xmlns:vor="http://wwwdbis.informatik.uni-kl.de/vorlesung">
    <vor:titel>Datenbankanwendung</vor:titel>
    <vor:thema>Inhalt und Motivation</vor:thema>
    <vor:thema>Anforderungen und Beschreibungsmodelle</vor:thema>
    <vor:thema>Logischer DB-Entwurf</vor:thema>
  </vor:vorlesung>
  <sem:seminar xmlns:sem="http://wwwdbis.informatik.uni-kl.de/seminar"
               sem:semester="WS 2002/03">
    <sem:titel>XML und Datenbanken</sem:titel>
    <sem:thema>
      <sem:name>XML-Grundlagen</sem:name>
      <sem:vortragender matnr="123456">Christian Weber</sem:vortragender>
      <sem:datum>24.01.2002</sem:datum>
      <sem:quelle>http://www.w3.org/XML/</sem:quelle>
      <sem:quelle>http://www.w3.org/TR/1998/REC-xml-19980210</sem:quelle>
    </sem:thema>
    <sem:thema>
      <sem:name>XML-Verarbeitungsmodelle und Language Bindings</sem:name>
      <sem:vortragender>Christian Müller</sem:vortragender>
      <sem:quelle>http://www.w3.org/XML/</sem:quelle>
    </sem:thema>
  </sem:seminar>
</lehrveranstaltungen>
```

Beispiel 8: Definition und Verwendung von Namensräumen

2.6 XML-Schema

Genauso wie mit einer DTD kann mit einem XML-Schema ([36]) eine Menge von Regeln definiert werden, mit denen ein XML-Dokument auf seine Gültigkeit hin überprüft werden kann. Ein XML-Schema-Dokument ist im Gegensatz zu einer DTD selbst auch ein XML-Dokument und kann, wie in Abbildung 4 gezeigt, mit einer DTD oder einem XML-Schema beschrieben werden.

Mit einer DTD kann der unstrukturierte Inhalt nicht bestimmten Datentypen zugeordnet werden. Es gibt nur den Datentyp #PCDATA. XML-Schema kennt 44 einfache Datentypen (z.B. string, int, date, float). Außerdem können neue Datentypen aus den einfachen Datentypen abgeleitet werden. Dadurch kann der Wertebereich eingeschränkt werden. Die abgeleiteten Datentypen können auch komplex sein, d.h., der neue Datentyp besteht aus mehren Teilen, denen verschiedene Datentypen zugeordnet sind. Beispiel 9 zeigt ein XML-Schema Dokument und einige der Möglichkeiten, mit denen man die Struktur eines XML-Dokumentes besser beschreiben kann. Durch die Verwendung eines XML-Schemas ist es

möglich mit einem Parser die Gütigkeit der Daten im XML-Dokument festzustellen. Wird ein XML-Dokument nur durch eine DTD beschrieben, so muss die Anwendung, die das Dokument verarbeitet, überprüfen, ob die Daten im Dokument gültig sind.

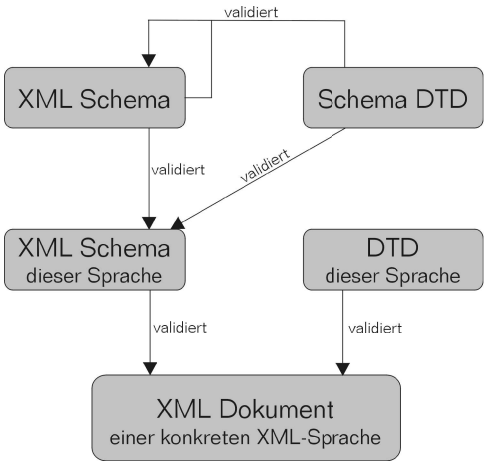

Abbildung 4: Beziehungen zwischen XML-Dokument, DTD und XML-Schema

```
<?xml version="1.0" encoding="iso8859-15"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:simpleType name="matType">
      <xs:restriction base="xs:int">
        <xs:minInclusive value="100000">
        <xs:maxExclusive value="1000000">
      </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="vortragenderType">
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="matNr" type="matType" use="implied"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>

    <xs:complexType name="themaType">
      <xs:sequence>
        <xs:element name="name"         type="xs:string"        minOccurs="1" maxOccurs="1" />
        <xs:element name="vortragender" type="vortragenderType" minOccurs="0" maxOccurs="1" />
        <xs:element name="datum"        type="xs:date"          minOccurs="0" maxOccurs="1" />
        <xs:element name="quelle"       type="xs:anyURI"        minOccurs="0"
                                                                maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>

    <xs:element name="seminar">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="titel" type="xs:string" minOccurs="1" />
          <xs:element name="thema" type="themaType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

  </xs:schema>
```

Beispiel 9: XML-Schema Dokument

3. Datenformat XML

XML hat sich zu einem universellen Speicherungs- und Austauschformat entwickelt. Die Speicherung von Daten im XML-Format benötigt mehr Systemressourcen und eine längere Laufzeit des Programms, da die Dokumente in jedem Verarbeitungsschritt komplett geparst werden müssen. Dieser Mehraufwand wird durch Vorteile wie z.B. Unabhängigkeit von Betriebsystemen und Programmiersprachen aufgewogen.

3.1 Parsen von XML-Dokumenten

Um auf die Daten von XML-Dokumenten zugreifen zu können oder ganze Dokumente zu verarbeiten, werden Anwendungen mit entsprechenden Schnittstellen benötigt, die wiederkehrende Aufgaben (z.B. das Parsen) erledigen. Die zwei bekanntesten Schnittstellen zum Parsen von XML sind das „Simple API for XML“ (SAX, [9]) und das „Document Object Modell“ (DOM, [10]). Der größte Unterschied liegt in der Verarbeitung der Daten des XML-Dokuments. SAX ist ereignisorientiert. Während das XML-Dokument gelesen wird werden keine Daten im Speicher abgelegt, da beim Lesen eines Elementes die Daten an eine in der Anwendung definierte Funktion übergeben werden. Mit Hilfe vom SAX können keine Daten modifiziert werden, da die Daten nur in dem Moment zur Verfügung stehen, in dem sie gelesen werden.
DOM ist objektorientiert. Beim Lesen des Dokuments wird ein Baum von Objekten im Speicher aufgebaut. Nachdem das gesamte Dokument gelesen wurde, kann es durch eine Anwendung beliebig geändert werden und anschließend wieder als Dokument geschrieben werden.

3.2 Verarbeitung von XML-Dokumenten

Auf Grundlage des XML-Standards wurden eine Reihe von Technologien entwickelt, die XML-Dokumente auf verschiedene Art und Weise verarbeiten. In Abbildung 5 ist das Verarbeitungsmodell von XML dargestellt.

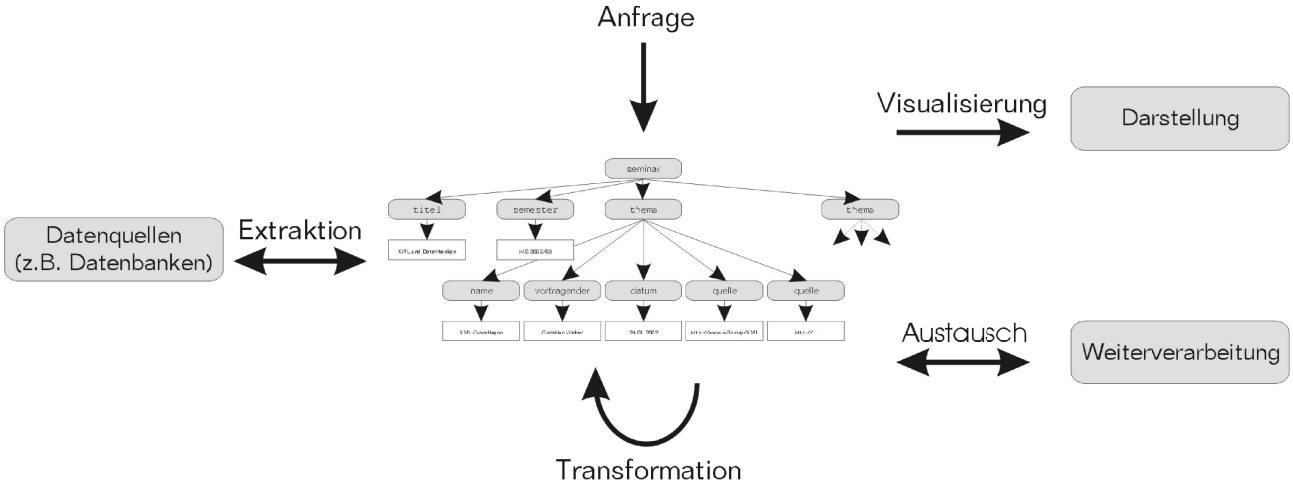

Abbildung 5: XML-Verarbeitungsmodell

Die Verarbeitung der Dokumente lässt sich in verschiedene Bereiche einteilen: Speicherung und Extraktion von XML-Dokumenten in und aus Datenbanken, Transformation in andere Dokumente, Austausch zur weiteren Verarbeitung, Aufbereitung zur Visualisierung sowie die Verarbeitung von Anfragen.

3.2.1 XML-Anfragesprachen

Eine XML-Anfragesprache dient als Werkzeug, um eine strukturierte und inhaltsbezogene Anfrage an eine oder mehrere Datenquellen zu stellen. Da XML-Dokumente anders strukturiert sind als Daten, die in relationalen oder objektorientierten Datenbanken gespeichert sind, können Anfragesprachen, die für Datenbanken entwickelt wurden (z.B. SQL [39]), nicht für Anfragen an XML-Dokumente verwendet werden. Besonderheiten in XML-Dokumenten (z.B. hierarchisches Datenmodell) können mit SQL nicht verarbeitet werden.

XPath [13], die XML Path Language, ermöglicht die flexible Adressierung von Teilen eines XML-Dokuments. Hauptsächlich wurde XPath entwickelt, um von anderen Technologien wie zum Beispiel XQuery [40], XSL [18] und XPointer [41] benutzt zu werden. XPath ermöglicht die Selektion einer Menge von Knoten. Beispiel 10 zeigt zwei XPath-Ausdrücken.

XPath-Ausdruck zur Lokalisierung aller Vortragender:
```
/seminar/thema/vortragender
```
Ergebnis:
```
<vortragender>Christian Weber</vortragender>
<vortragender>Christian Müller</vortragender>
```

XPath-Ausdruck zur Lokalisierung aller direkten Kindknoten des aktuellen Knotens:
```
/seminar/child::*
```
Ergebnisknotenmenge:
```
{titel, thema, thema}
```

Beispiel 10: XPath-Ausdrücke (in Bezug auf Beispiel 2)

XQuery ist eine Anfragesprache, die von der XML-Anfragesprache Quilt abgeleitet wurden. Dabei wurde die Pfadausdruckssyntax von XPath und die Variablenbindung (`<elem> $var </elem>`) von XML-QL (query language for XML ,[42]) übernommen. Klauseln mit Schlüsselwörtern wie SELECT – FROM – WHERE wurden bei SQL abgeschaut. XQuery arbeitet mit verschiedenen Ausdrücken, die beliebig verschachtelt sein dürfen. Als Ergebnis liefert ein Ausdruck immer eine Liste von Werten bzw. Knoten.

Es existieren noch viele andere semistrukturierte Abfragesprachen (zum Beispiel XML-QL, Lorel, UnQL, DQL, XML-GL, YATL), mit denen man die Daten aus XML-Dokumenten erhalten kann. Diese Sprachen sind alle an die geschachtelte Baumstruktur von XML angepasst worden und benutzen Pfad-Ausdrücke, um die gewünschten Daten zu erhalten.

3.2.2 XML und Datenbanken

Es existieren unterschiedliche Möglichkeiten für die Speicherung von XML-Dokumenten. Sie können im Dateisystem, in relationalen Datenbanken, in objektorientierten Datenbanken oder XML-enabled Datenbanken gespeichert werden. Dateien sind zur Speicherung von XML-Dokumenten sehr ungeeignet, da das Suchen auf einer Menge von Dateien nur sehr ineffizient durchgeführt werden kann. Wenn XML-Dokumente in einer Datenbank gespeichert werden, kann mit Hilfe von SQL-Anfragen sehr schnell gesucht werden. XML-Dokumente können recht einfach auf objektorientierte Datenbanken abgebildet werden und es existieren auch unterschiedliche Ansätze bei der Speicherung und Rekonstruktion von XML-Dokumenten in relationalen Datenbanken. Es gibt auch Systeme die auf die Speicherung von XML-Dokumenten speziell zugeschnitten sind.
Wenn XML-Dokumente in Datenbanken gespeichert und aus ihnen extrahiert werden sollen, ergeben sich Anforderungen an das Datenbanksystem, die von relationalen Datenbanksystemen nicht erfüllt werden. Mit Hilfe vorhandener Mechanismen müssen typische Eigenschaften von XML-Dokumenten

nachgebildet werden. Diese Anforderungen ergeben sich durch die unterschiedlichen Eigenschaften der Struktur des XML-Schema und des Datenbank-Schema, die nachfolgend erklärt sind:

- **XML-Dokumente enthalten ineinander verschachtelte Elemente.**
  Dabei ist die Verschachtelungstiefe nicht beschränkt. Bei relationalen Datenbanken müssen die verschachtelten Elemente zerlegt werden und mittels Fremdschlüsselbeziehungen verknüpft werden.
- **In XML-Dokumenten können Elemente innerhalb eines Elements beliebig oft wiederholt werden.**
  In relationalen Datenbanken ist eine Spalte immer nur einmal vorhanden und enthält einfache Werte. Wenn mehrere Elemente gespeichert werden müssen, ist eine zusätzliche Tabelle notwendig.
- **Ein Element eines bestimmten Typs in einem XML-Dokument muss nicht immer die gleichen Kindelemente enthalten.**
  In relationalen Datenbanken gibt es keine Möglichkeit von Alternativen.
- **In XML-Dokumenten ist die Reihefolge der Elemente ein fester Bestandteil der Dokumentstruktur.**
  In relationalen Datenbanksystemen spielt die Reihenfolge, in der die Werte gespeichert werden sollen, keine Rolle, d.h., die Reihenfolge der Elemente bleibt bei der Extraktion nicht zwingend erhalten. Es sind zusätzliche Tabellen nötig, um die Reihenfolgen der Elemente zu speichern.

Objekorientierte Datenbanksysteme wirken auf den Ersten Blick wie geschaffen für die Speicherung von hierarchisch strukturierten XML-Dokumenten, da sie grundsätzlich komplexe Datenstrukturen abbilden können. Sie sind jedoch auf die Speicherung von objektorientierten Programmen optimiert und können daher nicht effizient mit großen hierarchischen XML-Dokumenten umgehen. Eine Unterstützung der XML-Anfragesprache XPath ist nicht vorhanden.

Die Speicherung und Extraktion von XML-Dokumenten in relationalen oder objektorientierten Datenbanken hat einen großen Verarbeitungsaufwand zur Folge, besonders dann, wenn es sich um große oder komplexe XML-Dokumente handelt. Aus diesem Grund wurden neue Datenbanksysteme entwickelt, die speziell zur Speicherung von XML-Dokumenten gedacht sind. Diese Datenbanken, die die XML-Dokumente in ihrem nativen Format speichern, werden auch native XML-Datenbanken genannt. Wenn ein Datenbanksystem die XML-Dokumente nicht in ihrem nativen Format speichern, so wird diese Datenbank als XML-enabled Datenbank bezeichnet.

XML-Datenbanken zeichnen sich dadurch aus, dass die XML-Dokumente so gespeichert werden, dass der Zugriff auf Elementebene möglich ist, ohne dass immer das ganze Dokument gelesen oder gar geparst werden muss. Die im Abschnitt 3.1 beschriebenen XML-Anfragesprachen müssen verarbeitet werden können. Ergebnisse von Anfragen sollen in Form einer Menge von XML-Elementen bzw. einem temporärem XML-Dokument zur Verfügung gestellt werden. Bei der Speicherung soll das hierarchische Format des Dokuments erhalten bleiben.

3.2.3 Darstellung und Transformation von XML-Dokumenten

XML ist darstellungsneutral. Für die maschinelle Verarbeitung ist die Präsentation der Daten nebensächlich. Es gibt eine Trennung der Daten, die im Dokument gespeichert sind, von der Struktur, die durch eine DTD vorgegeben wird, und dem Layout zur Darstellung der Information.

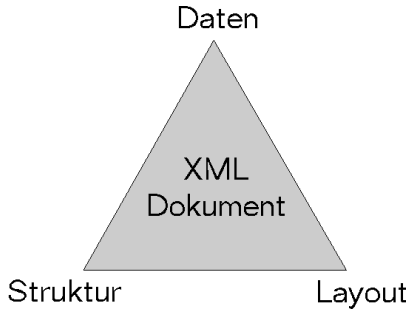

Abbildung 6: XML-Komponenten

In den folgenden Punkten werden verschiedene Möglichkeiten vorgestellt, um die in XML-Dokumenten gespeicherte Information darzustellen.

- Die einfachste Möglichkeit HTML-Dokumente zu formatieren, ist die direkte Darstellung eines mit der Cascading Stylesheet Language (CSS) [43] formatierten HTML-Dokuments in einem Browser. In einem CSS-Stylesheet wird für einzelne Elemente in einem HTML-Dokument angegeben, wie sie im Browser dargestellt werden sollen. XML Dokumente können CSS grundsätzlich verwenden, allerdings werden mit CSS nur ein Teil der Möglichkeiten von XML genutzt.
- Aus diesem Grund wurde eine eigene Formatierungssprache für XML entwickelt, die Extensible Stylesheet Language (XSL) [18]. Die Extensible Stylesheet Language besteht aus zwei Teilen: XSLT zur Transformation von XML-Dokumenten in eine andere Form (z. B. HTML) und den Formatting Objects (XSL-FO) zur Konvertierung in Printformate.

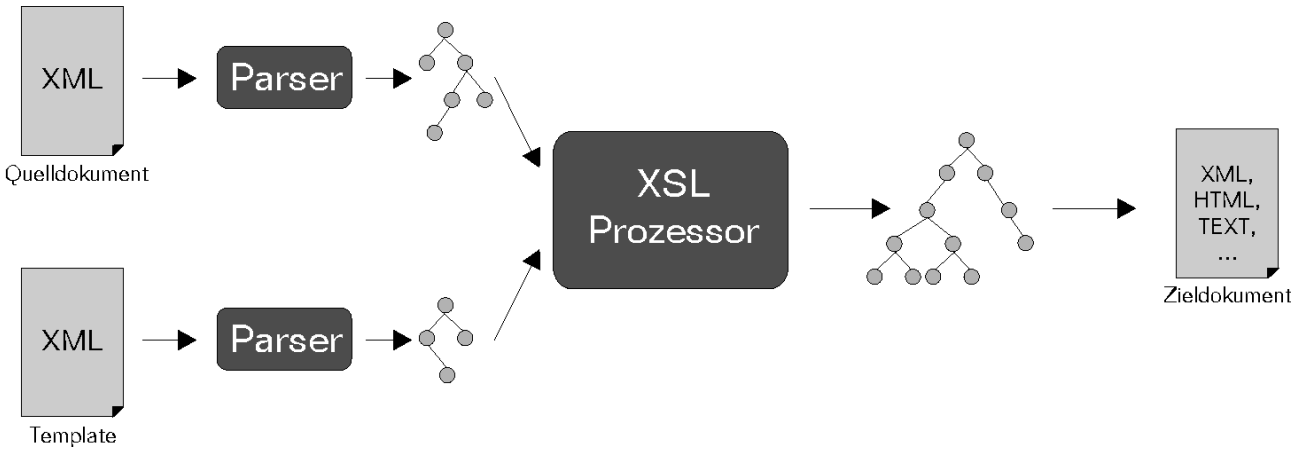

Abbildung 7: XSL-Arbeitsablauf

XSL Transformations (XSLT) [18] ist der wichtigste Teil von XSL. Es gibt eine Reihe von Regeln, mit denen ein neues Dokument beschrieben werden kann. Diese Regeln werden in Templates abgelegt. Diese Templates sind ebenfalls XML-Dokumente. Abbildung 7 zeigt den Arbeitsablauf der XSL Transformation. Zur Transformation durch den XSL-Prozessor wird das XML-Quelldokument sowie ein Template, in dem die Transformationsregeln aufgeführt sind, benötigt. Da der Transformationsprozess nicht direkt auf den Dokumenten arbeitet, sondern auf einer DOM-Repräsentation, müssen die Dokumente zunächst durch einen Parser in diese Darstellung überführt werden. Die Überführung in einen DOM-Baum ermöglicht es, dass zu jedem Zeitpunkt der Laufzeit auf alle Elemente und Daten zugegriffen werden kann, und das Dokument nicht sequentiell abgearbeitet werden muss.

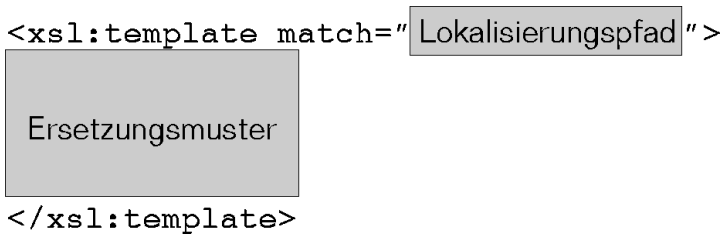

Abbildung 8: Transformationsschablone

Die Formatierungsanweisungen werden Template Rules genannt. Abbildung 8 zeigt eine entsprechende Transformationsschablone. Wenn der XSL-Prozessor auf eine Regel trifft, wird das gesamte XML-Dokument nach Elementen, die auf dieses Pattern passen durchsucht. Wird ein passendes Element gefunden, so wird das Template dem Ergebnisbaum hinzugefügt. Der gesamte Verarbeitungsprozess besteht aus einem Suchen von Elementen und dem Hinzufügen der entsprechenden Templates an den Ergebnisbaum. Um Daten in das Ergebnis zu übernehmen, müssen diese explizit angegeben und verarbeitet werden. Wenn für Elemente des bearbeiteten XML-Dokumentes keine Templates vorhanden sind, kommen sie in der Ausgabe auch nicht vor.

- XSL Formatting Objects (XSL-FO) [18] sind die dritte Möglichkeit der Präsentation von XML-Dokumenten. Ein XML-Dokument wird mit einem XSL-Stylesheet in ein XSL-FO-Dokument umgewandelt und dieses wird anschließend mit einem Formatierungsprogramm zum Beispiel in ein PDF-Dokument konvertiert. XSL-FO kennt 56 Elemente, mit denen wie bei TeX, PostScript oder PDF alles in Boxen dargestellt wird. Diese Boxen können nach Bedarf genau positioniert werden. Durch die seitenorientierte Verarbeitungsweise eignet sich XSL-FO auch für anspruchsvolle Textsatzaufgaben. Mit XSL-FO entfällt die Notwendigkeit, für verschiedene Ausgabemedien verschieden Umwandlungswerkzeuge einsetzen zu müssen.

3.2.4 XML WebServices

Von WebServices [23] wird derzeit häufig gesprochen. Die Definitionen des Begriffs WebService weichen oft voneinander ab. Einige sind sogar der Meinung, dass das, was wir heute als WebServices bezeichnen, nur die Vorstufe der eigentlichen WebServices sind.
WebService basieren auf eine Reihe von einzelnen Technologien. Dabei liegt der Focus auf Standards, die sich im Internet etabliert haben. Daher ist es auch nicht verwunderlich, dass viele der spezifischen Standards auf allgemeineren Standards wie XML beruhen.
Das bekannteste Protokoll für WebService ist sicher das Simple Object Access Protocol (SOAP) [29]. SOAP ist eine XML-basierte Sprache zur Verpackung von RPC-Aufrufen. Üblicherweise wird dabei HTTP als Transportprotokoll und das dort typische Request/Response-Muster unterstellt. SOAP definiert, wie ein Aufruf verpackt wird, wie das Ergebnis geliefert wird und wie ggf. Fehlermeldungen verpackt werden.

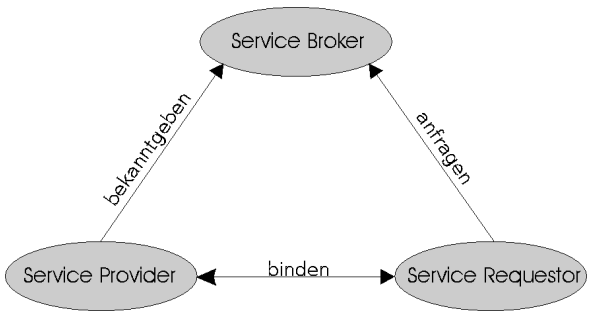

Abbildung 9: WebService Architektur

Abbildung 9 zeigt die Grobstruktur der WebServices. Aus Architektursicht werden WebService in der Regel als Dreigespann aus Provider, Requester und Broker dargestellt. Der Service-Broker arbeitet dabei als Vermittler zwischen Service-Requester und Service-Provider, damit diese sich finden können. Ein Service-Provider bietet WebServices an. Ein Service-Requester versucht mit Hilfe des Service-Brokers einen WebService zu finden. Wird ein WebService gefunden, so baut der Service-Requester eine Verbindung zum Service-Provider auf, um den WebService zu „konsumieren". Dabei ist es möglich, das der konsumierte WebService selbst wieder andere WebServices nutzt. Jeder dieser WebServices basiert auf der in Abbildung 8 dargestellten Architektur, so dass es sich um ein rekursives Modell handelt. Die Rekursion ist jedoch für den aufrufenden Services-Requester transparent.

Diese Sicht beschreibt nur sehr grob die Beziehungen zwischen den Beteiligten. Völlig offen lässt dieses Modell die interne Architektur. Weder wird ausgesagt, nach welchem Modell der Provider aufgebaut ist, noch, an welcher Stelle der Requester den Dienst in seiner Architektur ansiedelt.

Um einen WebService benutzen zu können, muss man zunächst wissen, dass dieser WebService existiert. Damit man den WebService ansprechen kann, benötigt man Information über seine Adresse und eine Schnittstellenbeschreibung. Auf diesem Gebiet ist zur Zeit UDDI (Universal Description, Discovery and Integration) [44] der bekannteste zentrale Verzeichnisdienst. Wegen seiner Arbeitsweise wird er auch als Gelbe Seiten für WebService bezeichnet. UDDI ist im wesentlichen eine Datenbank mit genau den notwendigen Informationen um einen WebService zu finden.

Um einen Aufruf an eine Komponente zu machen, muss man dessen Schnittstelle kennen. Welche Aufrufe können gemacht werden, welche Parameter müssen übergeben werden, wohin muß der Aufruf geschickt werden, etc. sind die Fragen, die diese Beschreibung beantworten muß. Im WebService-Umfeld wird diese Aufgabe durch die WebService Description Language (WSDL) [45] erfüllt.

Aus Sicht der verwendeten Technologien ist es derzeit so, dass das Gespann SOAP, WSDL und UDDI gemeinhin als Kern des Begriffs WebService gilt. Problembereiche wie verlässliche Kommunikation oder Sicherheit werden als offene Flanken akzeptiert, die derzeit durch proprietäre Technologien aufgefüllt werden.

3.2.5 XML Data-Binding

XML Data-Binding beschäftigt sich mit der Transformation von XML-Dokumenten in das Objektmodell einer Programmiersprache (z.B. Java) und umgekehrt. Mit Hilfe dieser Technologie ist es für den Programmierer einer Anwendung wesentlich einfacher die Daten zu verändern.

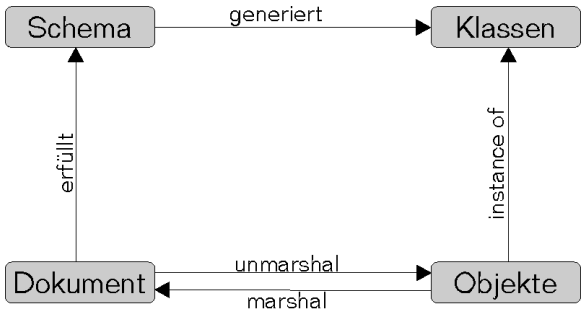

Abbildung 10: XML Data-Binding Struktur

Die XML Data Binding Produkte lassen sich in zwei verschiedene Kategorien einteilen. Produkte, die zur Entwicklungszeit einer Anwendung konfiguriert werden müssen, und Produkte, die keine

Konfiguration benötigen und direkt im Code verwendet werden können.

Die Produkte, die eine Konfiguration erfordern, sind flexibler in Bezug auf ein Mapping zwischen XML-Dokumenten und Objekten. Diese Produkte unterstützen zwei verschiedene Arten der Verwendung. Entweder kann der Programmierer ein Mapping zwischen XML-Elementen und Objekten definieren oder mit Hilfe eines XML-Schemas oder einer DTD entsprechende Klassen generieren lassen. Abbildung 10 zeigt den Zusammenhang zwischen Dokumenten, Schema, Klassen und Objekten.

Bei der Verwendung von Data Binding Produkten, die dynamisch zur Laufzeit eingesetzt werden können, kann der Entwickler keinen Einfluß auf die Verarbeitungsweise nehmen. Dem Entwickler einer Anwendung kann die Namensgebung und die Struktur der erstellten XML-Datei völlig egal sein, wenn XML als Speicherungsformat für Konfigurationsdaten einer Anwendung genutzt wird, die von keiner anderen Anwendung verarbeitet werden sollen.

Konzeptionell entwickelt der Data Binding Ansatz die bereits mit dem Document Object Model verwirklichte Grundidee einer Speicherabbildung von XML-Dokumenten fort. Für den Entwickler einer Anwendung ist die Verwendung von XML Data Binding Produkten sehr vorteilhaft, da er auf diese Weise kaum noch mit XML in Berührung kommt.

4. Zusammenfassung und Ausblick

Mit dieser Einführung in die eXtensible Markup Language wurde der grundlegende Aufbau und die Vorteile von XML als Speicherungs- und Austauschformat dargestellt. Die grundlegenden Möglichkeiten der Verarbeitung und des Austauschs von XML-Dokumenten wurden mit Hilfe des XML-Verarbeitungsmodells aufgezeigt.

Die nachfolgenden Vorträge, die sich in die Themengebiete Anfrageverarbeitung bzw. -formulierung, Datenbank-orientierte Verarbeitung, Dokumenten-orientierte Verarbeitung und XML-basierte Integrationskonzepte einteilen lassen, werden einige der hier angesprochenen Technologien aufgreifen und vertiefen.

Literatur:

1. Elliotte Rusty Harold, XML Bibel, IDG Books Worldwide, Inc. 1999
2. Elliotte Rusty Harold & W.Scott Means, XML in a nutshell, O'Reilly Verlag 2001
3. Brett McLaughlin, Java und XML, O'Reilly Verlag 2000
4. W3C, XML
   http://www.w3c.org/XML/
5. Veikko Wünsche, Einführung in XML
   http://www.lightwerk.com/seminars/xml/xml/xml-intro.pdf
6. Bärbel Bornemann Janine Czubak Klaus Knörzer, XML - Teil 1 / Grundlagen
   http://www.stud.fernuni-hagen.de/q3329747/
7. Erik Wilde, XML - Grundlagen, Prinzipien und Anwendungen
   http://dret.net/lectures/xml-ss02/
8. Prof. Dr. H.-J. Schek, Dr. K. Böhm, Interoperable Informationssysteme
   http://www.dbs.ethz.ch/~ii/SS2001/index.html
9. Simple API for XML (SAX)
   http://www.saxproject.org/
10. Document Object Model (DOM)
    http://www.w3.org/DOM/
11. Manuel Roellinghoff, XML-Parser
    http://www.fh-wedel.de/~si/seminare/ws00/Ausarbeitung/7.xmlparser/index.html
12. Robert Wruck, XLink, XPath & Xpointer
    http://www.fh-wedel.de/~si/seminare/ws00/Ausarbeitung/3.xlink/xlink0.htm
13. W3C, XPath
    http://www.w3c.org/TR/xpath
14. Angelika Müller, XML-Anfragesprachen
    http://www3.informatik.tu-muenchen.de/lehre/SS2001/HSEM-bayer/ausarbeitung3.pdf
15. Datenbank und das Web
    http://herakles.fzi.de/dbxml/frame_set/frame_haupt_folien.htm
16. Hicham Loukili, XML-Systeme mit Schwerpunkt "native" XML-DBMS
    http://www3.informatik.tu-muenchen.de/lehre/SS2001/HSEM-bayer/ausarbeitung9.pdf
17. Sormaz Ümit, XML und Datenbanken
    http://wwwagss.informatik.uni-kl.de/Lehre/Proseminar02/Ausarbeitung/11/index.org.htm
18. W3C, XSL and XSLT
    http://www.w3c.org/Style/XSL/
19. Dominic Giger, XSL zur flexiblen Darstellung von XML-Dokumenten
    http://www2-iiuf.unifr.ch/is/PDF/SA_Giger_2002.pdf
20. Christian Gieche & Florian Weitling, Die eXtensible Stylesheet Language
    http://www.informatik.uni-freiburg.de/proglang/teaching/ss2001/xml/XSL.pdf
21. Roger Rosette, XSLT
    http://www3.informatik.tu-muenchen.de/lehre/SS2001/HSEM-bayer/ausarbeitung7.pdf
22. Heiko Faasch, XSLT- Die XSL Transformationssprache
    http://www.fh-wedel.de/~si/seminare/ws00/ Ausarbeitung/5.xslt/xslt0.htm
23. W3C, WebServices
    http://www.w3c.org/2002/ws/
24. Mario Jeckle, WebServices
    http://www.jeckle.de/webServices
25. WebServices
    http://www.webservices.org
26. Sander Duivestein, Web Services and Workflow
    http://www.webservicesarchitect.com/content/articles/sander01.asp

27. Mario Jeckle, WebServices
http://www.studentconsultant.org/germany/ augsburg/files/WebServices.pdf
28. Alexander Jung, Links zum Thema WebServices
http://www.newline-people.net/alexander.jung/tech/ws-links.htm
29. W3C, SOAP
http://www.w3c.org/2000/xp/Group/
30. SOAP - Einführung, FEI QI
http://trumpf-3.rz.uni-mannheim.de/www/sem2002s/fei/SOAP-1.htmlOAP-1.html
31. Ronald Bourret, XML Data Binding Resources
http://www.rpbourret.com/xml/XMLDataBinding.htm
32. Ming Zha und Thorsten Seck, Java & XML Data Binding
http://www.st.informatik.tu-darmstadt.de:8080/~haupt/lehre/xml/13f.pdf
33. Orientation in Objects GmbH, XML-Data-Binding
http://www.oio.de/m/xmlbased/schema-klassen-objekte-dokument.htm
34. W3C, HyperText Markup Language (HTML)
http://www.w3.org/MarkUp/
35. W3C, Overview of SGML Resources
http://www.w3.org/MarkUp/SGML/
36. W3C, XML Schema
http://www.w3.org/XML/Schema
37. PHP: Hypertext Preprocessor
http://www.php.net
38. Uniform Resource Identifiers (URI) Working Group
http://ftp.ics.uci.edu/pub/ietf/uri/
39. SQL.ORG
http://www.sql.org
40. W3C, XQuery
http://www.w3.org/XML/Query
41. W3C, XPointer
http://www.w3.org/XML/Linking
42. W3C, XML-QL: A Query Language for XML
http://www.w3.org/TR/NOTE-xml-ql/
43. W3C, Cascading Stylesheet Language
http://www.w3.org/Style/CSS/
44. Universal Description, Discovery and Integratrion
http://www.uddi.org/
45. W3C, Web Services Description Language (WSDL)
http://www.w3.org/TR/wsdl