XML: Repräsentation von Struktur

Seminar: XML und Datenbanken
Universität Kaiserslautern WS02/03

Jochen Tuchbreiter

27. Januar 2003

Zusammenfassung

Im Kielwasser des Internets traten auch Auszeichnungssprachen ihren Siegeszug an: Zuerst mit der Einführung des WWW in Form des bekannten HTML. Nach einer Rückbesinnung auf die Mutter aller Auszeichnungssprachen, der „Standard Generalized Markup Language“ SGML, entstand 1996 der Neuentwurf „eXtensible Markup Language“ (XML). Hierbei handelt es sich im Wesentlichen um eine leichtere Version der sehr umfangreichen SGML Spezifikation.

Inzwischen existiert eine reichhaltige Menge an Standards, welche auf XML aufsetzen: So zum Beispiel XPath, eine Sprache zur Adressierung von Teilen von XML-Dokumenten, XSLT, ein Standard zur Transformation von XML-Dokumenten in andere Formate und auch XQuery, eine Abfragesprache für XML Daten.

Diese Ausarbeitung beschäftigt sich aber mit einer weiteren Klasse von Sprachen, nämlich solche mit deren Hilfe sich die Struktur von XML-Dokumenten beschreiben lässt. Im Rahmen der Ausarbeitung wird deutlich gemacht worin der Nutzen solcher Sprachen besteht und wo sie angewendet werden. Weiterhin werden die beiden wichtisten Vertreter dieser Art, „Document Type Defintions“ und „XML Schema“ näher erklärt und untersucht. Zum Abschluss werden weniger relevante aber dennoch interessante Ansätze zur Strukturbeschreibung vorgestellt.

Inhaltsverzeichnis

1 Einführung

Im Folgenden soll dargelegt werden welche Relevanz das Thema Repräsentation von Struktur in XML besitzt.

1.1 Gründe für den Einsatz von Strukturbeschreibungen

Da XML eine Meta-Sprache ist kann sie grundsätzlich für sehr verschiedene Zwecke eingesetzt werden. Unabhängig von dem tatsächlichen Einsatzzweck gilt es jedoch Kriterien dafür festzulegen wie ein erwünschtes und wie ein unerwünschtes XML-Dokument aussehen sollte. Im XML-Standard [1] sind dafür zwei Kriterien vorgesehen:

1. Wohlgeformtheit

   Ein XML-Dokument heißt wohlgeformt, wenn es genau ein Wurzel-Element enthält und jedes öffnende Tag auf der gleichen Ebene auch wieder geschlossen wird. Dies bedeutet, dass die Elemente so angeordnet sein müssen, dass sie in jedem Fall eine Baumstruktur repräsentieren, wobei der Baum genau einen Wurzelknoten besitzt.

2. Gültigkeit

   Ein XML-Dokument ist dann gültig (valid) wenn es eine bestimmte Struktur einhält. Wie diese Struktur im Einzelnen aussieht wird durch eine Strukturbeschreibung festgelegt. Strukturbeschreibungen sind das Kernthema dieser Ausarbeitung.

Ist ein XML-Dokument wohlgeformt, so ist es auf jeden Fall syntaktisch analysierbar und als Baum darstellbar. Oft ist dies jedoch nicht ausreichend: Wird XML beispielsweise für den Datenaustausch zwischen Systemen verschiedener Hersteller verwendet, so müssen diese Systeme das Dokument nicht nur syntaktisch analysieren, sondern auch interpretieren können. Um diese Interpretierbarkeit zu garantieren, muss jedoch festgelegt sein welche Elemente mit welchen Namen und Attributen in welcher Reihenfolge auftreten können.

Grundsätzlich wäre es denkbar, dass die Form dieser Festlegung den verschiedenen Parteien, welche eine gemeinsame Klasse von XML-Dokumenten verwenden möchten, überlassen bleibt. In XML jedoch wird dies durch eine formale Strukturbeschreibung in einem standardisierten Format realisiert. Dabei wirkt diese wie ein Vertrag zwischen den verschiedenen Nutzern eines Dokuments: Jedes System kann sich darauf verlassen dass es nur XML-Dokumente verarbeiten muss, welche diesem Vertrag entsprechen.

Eine Verankerung der Strukturbeschreibung in XML hat den Vorteil, dass nicht mehr jeder Nutzer eines XML-Dokumentes seine eigene Implementation eines Gültigkeitsprüfers schreiben muss, sondern sich alle Nutzer auf wenige standardkonforme Implementationen stützen können. Dies und die formale Festlegung der Struktur mit Hilfe einer Strukturbeschreibungssprache spart Aufwand und verhindert Konflikte in Bezug auf die Gültigkeit von Dokumenten.

1.2 Anwendungsmöglichkeiten

Der Einsatz einer Strukturbeschreibungssprache ist bei allen Klassen von XML-Dokumenten, welche von mehr als einer Partei genutzt werden, stark zu empfehlen. Im Folgenden werden hierfür einige Beispiele geschildert:

1. Kommunikation im B2B-Umfeld

   Im Bereich der Business-to-Business Kommunikation hat die elektronische Kommunikation eine besonders große Bedeutung, da hier beispielsweise zwischen Hersteller und Zulieferer oft zahlreiche Kommunikationsvorgänge mit großer Geschwindigkeit abgewickelt werden müssen. XML ist in diesem Gebiet inzwischen neben älteren Standards fest etabliert. Die Einführung von branchenspezifischen XML-Dialekten mit ihren zugehörigen Strukturbeschreibungen bietet sich an und wird bereits durchgeführt. Beispielsweise findet sich unter [2] eine Sammlung von Strukturbeschreibungen verschiedener XML-Dialekte.

2. Archivierung in digitalen Bibliotheken

   Bei der Archivierung von Inhalten in digitale Bibliotheken ist es besonders wichtig, dass diese über einen großen Zeitraum verfügbar und von der verwendeten Software unabhängig bleiben. Bei der Erfüllung dieser Anforderungen hilft eine formale Definition des Formats in welchem die Daten abgelegt wurden durch eine Strukturbeschreibung. Weiterhin nützt die Verwendung von gemeinsamen Strukturen auch bei der Vernetzung digitaler Bibliotheken. Beispielsweise wurde in einem von der Deutschen Forschungsgesellschaft geförderten Projekt eine Strukturbeschreibung für Dissertationen entwickelt [3].

2 Document Type Definitions

2.1 Konzepte und Geschichte von DTDs

Document Type Definitions (kurz: DTDs) sind der in XML 1.0 [1] spezifizierte Weg zur Beschreibung von Struktur von Dokumentklassen. Sie sind abgeleitet von den bereits länger vorhandenen DTDs der allgemeinen Markupsprache SGML und wurden auf XML zugeschnitten. Allerdings können DTDs ihre SGML-Herkunft nicht verbergen: Ihre Sprachkonstrukte sind - wie auch SGML selbst - klar dokumentzentriert und auch ihre Syntax ist nicht wohlgeformt und entspricht somit nicht derer der anderen XML-Standards wie beispielsweise XSLT. Viele der Einschränkungen von DTDs ergeben sich aus ihrer Herkunft.

Mit Hilfe von DTDs lassen sich Regeln zum Aufbau eines XML-Dokuments definieren: Es werden Angaben über die Namen, Anordnung und Reihenfolge von gültigen Tags gemacht, sowie deren Attribute und eventuelle Default-Werte festgelegt. Außerdem bieten DTDs Mechanismen zur Ersetzung von Text in XML-Dokumenten, sowie zur Aufnahme von nicht-XML Inhalten. Ein weiteres Merkmal von DTDs ist, dass sie Referenzen eines Elements auf ein anderes ermöglichen.

2.2 Sprachkonstrukte

Da eine ausführliche Beschreibung aller Sprachkonstrukte und Möglichkeiten von DTDs den Rahmen dieser Ausarbeitung sprengen würde beschränkt sich der Autor auf die wichtigsten Aspekte. Für eine ausführliche Beschreibung siehe [4].

2.2.1 Integration in XML-Dokumente

Die Integration von DTDs in ein XML-Dokument erfolgt über eine spezielle Anweisung: DOCTYPE. Mit Hilfe dieser Anweisung ist es sowohl möglich DTD Regeln direkt im Dokument anzugeben als auch über URLs auf externe Ressourcen zu verweisen, welche die für dieses Dokument anzuwendenden Regeln enthalten. Mit Hilfe der DOCTYPE Anweisung kann maximal ein externer DTD integriert werden.

Wird in der DOCTYPE Anweisung sowohl ein externer DTD angegeben als auch dokumentinterne Regeln verwendet so werden letztere zuerst ausgewertet und haben bei Namenskollisionen Vorrang vor den im externen DTD gemachten Angaben.

Beispiel:

```
<!DOCTYPE motor SYSTEM "file://motor.dtd" >
```

Dies legt fest dass die in der Datei motor.dtd festgeschriebenen Regeln für dieses XML-Dokument angewendet werden sollen. Dabei muss das Wurzeltag des XML-Dokuments <motor> heissen.

2.2.2 Festlegen gültiger Elemente

Um die Struktur von XML Dokumenten zu beschreiben bedarf es einer Möglichkeit auszudrücken welche Elemente in welcher Reihenfolge und Schachtelung gültig sind. DTDs verwenden hierzu ELEMENT Deklarationen. Hier gibt es zwei verschiedene Formen:

1. Kategorie des Inhaltes beschreibende Deklarationen

   `Syntax: <!ELEMENT name Inhaltskategorie >`

   Diese Deklarationen legen fest zu welcher Kategorie die Daten zwischen den öffnenden und schließenden Tags mit dem Namen „name“ gehören müssen. Mögliche Kategorien sind:

   - Any - Jegliche wohlgeformten Daten oder
   - None - überhaupt keine Daten oder Elemente

2. Modell des Inhalts beschreibende Deklarationen

   `Syntax: <!ELEMENT name (Inhaltsmodell) >`

   Mit Hilfe dieser Deklarationen können die zwischen dem durch „name“ bezeichneten öffnenden und schließenden Tag vorkommenden Daten einem Inhaltsmodell zugeordnet werden. Mögliche Modelltypen sind:

   - #PCDATA
     Element enthält nur Text.
   - elementname1, elementname2, elementname3, ...
     Element enthält weiteres Element mit dem Namen elementname1 gefolgt von einem mit dem Namen elementname2, gefolgt von einem mit dem Namen elementname3 ... .

   Modelltypen können auch Alternativen enthalten, welche mit dem Zeichen „|“ getrennt werden. So legt

   `<!ELEMENT foo (elementname1 | elementname2) >`

   fest, dass im Element foo entweder ein Element mit dem Namen elementname1 oder eines mit dem Namen elementname2 vorkommen muss.

   Neben Alternativen können auch Multiplizitäten angegeben werden. Dies geschieht durch Anfügen eines Multiplizitätsoperators. Gültige Multiplizitätsoperatoren sind:

- keiner: 1 Vorkommen
- ?: 0-1 Vorkommen
- +: 1-n Vorkommen
- *: 0-n Vorkommen

Ein Beispiel hierfür:

```
<!ELEMENT foo (elementname1?) >
```

Dies legt fest, dass das Element foo entweder gar keine Daten oder ein Element mit dem Namen „elementname1“ enthält.

Mit Hilfe dieser zusätzlichen Möglichkeiten kann auch festgelegt werden dass ein Element eine Mischung aus Text und anderen Elementen enthalten muss, wie es z.B. bei XHTML Anwendung findet:

```
<!ELEMENT foo (#PCDATA | elementname1 | elementname2 | ...)* >
```

Diese Deklaration sagt aus, dass das Element foo eine Mischung aus Text und aus den Elementen elementname1, elementname2, ... enthalten kann. Gültig wäre also beispielsweise:

```
<foo>
  Dies ist ein Text.
  <elementname2/>
  Dies ist ein weiterer Text
  <elementname1/>
  Dies ist noch ein weiterer Text
</foo>
```

2.2.3 Festlegen gültiger Element-Attribute

Neben der Schachtelung erlaubt XML auch die Zuweisung von Attributen zu einzelnen Elementen. Um die Struktur eines XML-Dokumentes zu beschreiben, müssen also auch gültige Attributnamen und Werte beschrieben werden. Im Falle von DTDs geschieht dies über ATTLIST-Deklarationen, welche syntaktisch folgendermassen aufgebaut sind:

```
<!ATTLIST elementName
  attrName1 attrTyp1 attrVorgabe1 attrStandardWert1
  attrName2 attrTyp2 attrVorgabe2 attrStandardWert2
  ... >
```

Hierbei bezeichnet „elementName“ den Namen des Elements für welches die Attributbeschreibung gelten soll. Dieser wird gefolgt von einer Reihe von Beschreibungen der einzelnen Attribute des Elements: „attrName“ beschreibt den Namen des Attributs, „attrTyp“ den Typ

gültiger Werte gefolgt von „attrVorgabe“, welches festlegt ob ein Attribut für das angegebene Element auf jeden Fall vorhanden sein muss. Zuletzt wird durch attrStandardWert der Standardwert des Attributes festgelegt. Dieser findet Anwendung wenn, dem Attribut kein anderer Wert zugewiesen wurde. DTDs definieren zehn mögliche Attributtypen. Die Wichtigsten:

- CDATA

  Beschreibt beliebigen Text.

- Wert-Aufzählungen

  Beschreibt gültige Werte anhand einer Aufzählung. Jegliche nicht in der Aufzählung enthaltenen Werte sind ungültig.

- ID

  Beschreibt einen eindeutigen Schlüssel für jede Instanz des Elements. Dies entspricht grob dem Konzept der Primärschlüssel bei Datenbanken.

- IDREF

  Beschreibt eine Referenz auf ein Element mit einem ID Attribut. Dies entspricht grob dem Konzept eines Fremdschlüssels bei Datenbanken. Mit Hilfe dieses Typs können einmal definierte Elemente später im XML-Dokument referenziert werden und müssen somit nicht mehrfach vorkommen.

- IDREFS

  Beschreibt eine Menge von Referenzen auf Elemente mit einem ID Attribut. Mit Hilfe dieses Typs lassen sich n:m Verbindungen zwischen Elementmengen aufbauen.

- ENTITIY

  Beschreibt den Namen einer vorderfinierten Einheit (siehe Kapitel „Weitergehende Definitionen“).

Mögliche Werte für Attributvorgaben sind:

- #REQUIRED

  Attribut muss immer angegeben werden.

- #IMLPIED

  Attribut ist optional.

- #FIXED

  Attribut ist optional. Falls es aber angegeben wird muss es den Standardwert enthalten. Falls nicht wird es durch einen validierenden Parser automatisch auf den Standardwert gesetzt.

- Standardwert

  Wird keines der obigen Schlüsselwörter angegeben so wird die Angabe als Standardwert interpretiert. Falls das Attribut nicht angegeben wird, wird ihm von einem validierenden Parser dieser Standardwert zugewiesen. Falls es aber angegeben wird kann es einen beliebigen - zum Attributtyp konformen - Wert haben.

Ein Beispiel für eine Attributliste für ein Elmenet foo, welches alle vorgestellten Attributvorgaben nutzt:

```
<!ATTRLIST foo
  attr1 ID #REQUIRED
  attr2 CDATA #IMPLIED
  attr3 CDATA #FIXED "hello"
  attr4 CDATA "hello" >
```

2.2.4 Weitergehende Definitionen

Obwohl es mit den bisher beschriebenen Sprachelementen möglich ist Struktur von XML-Dokumenten umfassend zu beschreiben, bieten DTDs weitergehende Möglichkeiten an. Die ENTITIY Deklaration eignet sich sowohl zur Verkürzung von XML-Dokumenten und DTDs als auch zur Erhöhung der Flexibilität selbiger. Weiterhin bietet sie die Möglichkeit nicht-XML Daten in XML-Dokumente einzubinden. Entities können als eine Art Makro im Sinne der Präcompiler Makros der Sprache C betrachtet werden: Es geht hier vornehmlich um Textersetzung.

Dabei wird unterschieden zwischen Entities, welche in XML Daten Anwendung finden („General Entities“) und denen die in DTDs selbst genutzt werden („Parameter Entities“). Weiterhin wird zwischen Entities, welche wohlgeformten Inhalt haben müssen und solchen die beliebigen Inhalt haben können unterschieden.

Mit Hilfe von wohlgeformten General Entities kann während der Gültigkeitsprüfung Textersetzung in XML-Dokumenten durchgeführt werden: Jede im Dokument gefundene Referenz auf eine vorher definierte Entity wird durch deren Inhalt ersetzt. Referenzen haben die Form „&name;“ wobei „name“ der Name der zu referenzierenden Entitiy ist. Bekannt sind solche Referenzen aus XHTML: Hier werden Sonderzeichen wie Umlaute mittels Entity-Referenzen eingebunden. Solche Entities werden wie folgt deklariert:

```
<!ENTITY name "inhalt">
```

Beispiel:

```
<!ENTITY greeting  "Herzlich Wilkommen">
```

Eine solche Deklaration würde folglich jedes Vorkommen von &greeting; im XML-Dokument durch "Herzlich Wilkommen" ersetzen. Diese Funktionalität lässt sich beispielsweise zur Internationalisierung der Daten von XML-Dokumenten einsetzen.

Eine ähnliche Ersetzung ist mit Hilfe von wohlgeformten Parameter Entities auch in DTDs selbst möglich. Hier ist die Syntax leicht unterschiedlich: Referenzen werden mit „%name;" erzeugt. Solche Entities können genutzt werden, um oft wiederkehrende Deklarationen in DTDs nur einmalig als Entity zu deklarieren und damit DTDs kürzer und übersichtlicher zu gestalten. Eine weitere Anwendung ist der Einsatz im Zusammenhang mit bedingten DTD-Teilen: Hier können Entity-Deklarationen dafür sorgen dass bestimmte Teile eines DTDs genutzt und andere ausgeblendet werden. Macht man sich zusätzlich zunutze, dass interne DTDs vor externen DTDs ausgewertet werden, so ist es möglich im XML-Dokument den verwendeten externen DTD durch Entity-Deklarationen im internen DTD - also im XML-Dokument selbst - anzupassen.

2.3 Grenzen von DTDs

In der Praxis ergeben sich bei der Verwendung von DTDs einige Problemfelder. Die Wichtigsten sind im Folgenen aufgelistet:

1. Sehr schwache Typisierung

   DTDs erlauben nur sehr schwache Angaben über die Typen von einzelnen Attributen oder Elementinhalten: Die einzig praktikabel wählbaren Typen sind Freitext in verschiedenen Ausprägungen oder eine Angabe von festen Alternativen. Selbst primitive Typen beispielsweise zur Bezeichnung von Zahlen fehlen völlig. Weiterhin fehlt jegliche Unterstützung für zusammengesetzte Typen.

2. Anzahl der DTDs pro Dokument ist auf eins beschränkt

   Diese Beschränkung macht es schwierig DTDs modular aufzubauen und Dokumentklassen aus einzelnen Modulen zusammenzusetzen. Zwar lässt sich diese Einschränkung durch den Einsatz von externen Entities weitgehend umgehen, doch beruht diese Technik auf einem Seiteneffekt und verkompliziert die Struktur der Dokumente unnötig.

   Beim Einsatz dieser Technik treten umgehend weitere Probleme auf: Durch die mangelnde Unterstützung von Namespaces in

DTDs ist nur eine Kombination von DTD-Modulen mit disjunkten Elementnamen möglich. Während dies in einer firmeninternen Sammlung von Modulen noch tragbar sein mag, verursacht es spätestens beim Aufbau von Repositories aus weltweit standardisierten Modulen große Probleme.

3. Fehlende Unterstützung von Erweiterung/Vererbung von DTDs

   DTDs sehen keinen Mechanismus zur Erweiterung bzw. Vererbung vor. Dies verschlechtert die Chance der Wiederverwendung bereits geschriebener DTDs.

4. Fehlende Wirksamkeitsgarantie von externen DTDs

   Dokumente können in ihrem internen DTD alle im externen DTD festgelegten Regeln beliebig überschreiben. Daher ist die Aussage, dass ein Dokument einen bestimmten externen DTD verwendet und gültig ist, allein oft nicht ausreichend: Sie garantiert nicht dass sich das Dokument auch von auf den externen DTD zugeschnittenen Applikationen verarbeiten lässt.

3 XML Schema

3.1 Konzepte und Geschichte von XML Schema

Aufgrund der Unzulänglichkeiten von DTDs, welche zu großen Teilen mit ihrer SGML-Herkunft zu begründen sind, besteht Bedarf nach einer neuen Strukturbeschreibungssprache. Der im Folgenden vorgestellte Ansatz XML Schema löst viele der mit DTDs nicht bewältigbare Probleme. Er hat seid dem 02.05.01 den Status einer W3C Recommendation und liegt derzeit in der Version 1.1 vor. Die Spezifikation besteht aus drei Teilen: Teil 1 [6] ist eine Einführung , Teil 2 [7] beschreibt Sprachkonstrukte, während Teil 3 [8] näher auf Datentypen eingeht.

Die Syntax von XML Schema ist XML konform. Dies hat zur Folge, dass Software zur Validierung von XML-Dokumenten, deren Struktur mit Hilfe von XML Schema beschrieben ist, keinen separaten Parser für die Strukturbeschreibung mehr enthalten muss, sondern den sowieso vorhandenen XML Parser einsetzen kann. Weiterhin können XML Schema Definitionen somit in das bekannte „Document Object Model“ [9] abgebildet werden.

XML Schema erlaubt eine starke Typisierung von Elementen und Attributen, sowie die Konstruktion eigener Typen. Weiterhin wird auch nachträgliche Erweiterung von Schema Dokumenten unterstützt.

3.2 Sprachkonstrukte

3.2.1 Integration in XML-Dokumente

Zur Integration von XML Schema Dokumenten in eine Dokumentinstanz werden Attribute des XML-Schema Namensraumes [10] selbst genutzt. Aus diesem Grund müssen Dokumentinstanzen den XML Schema Namespace auch deklarieren. Dies kann wie in Beispiel 3.2.1.1 aussehen.

```
<rootElement
  xmlns="nameSpaceOfDocument"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://somewhere.com/schema schema.xsd
                      http://somewhere.com/schema2 schema2.xsd">
```

Beispiel 3.2.1.1

Hier wird deklariert, dass die beiden XML Schema Dokumente schema.xsd und schema2.xsd zur Validierung der Dokumentinstanz genutzt werden sollen. Der Einsatz von schemaLocation setzt jedoch vorraus, dass das eingebundene Schemadokument einen targetNamespace deklariert. Schemata, welche dies nicht tun, werden über noNameSpaceSchemaLocation="uri" eingebunden.

XML Schema erlaubt im Gegensatz zu DTDs die Einbindung beliebig vieler Strukturbeschreibungsdokumente in die Dokumentinstanz. Weiterhin ist es auch möglich, dass XML Schema Dokumente sich gegenseitig nutzen und erweitern.

Grundsätzlich ist ein validierender Parser jedoch nicht gezwungen die im Dokument angegebenen Schema-Dokumente zur Validierung zu nutzen, sondern er kann die zu nutzenden Dokumente auch auf anderen Wegen ausfindig machen.

Ähnlich wie bei DTDs gibt es auch für XML Schema Sprachkonstrukte zur Definition von gültigen Elementen und ihren Attributen. Darüber hinausgehend existieren jedoch auch solche, welche die Festlegung von Typen erlauben. Aufgrund der hohen Komplexität und Flexibilität von XML Schema nimmt eine ausführliche Sprachbeschreibung weitaus mehr Platz ein als hier zur Verfügung steht. Daher wird hier nur auf einzelne Aspekte eingegangen, zur genaueren Betrachtung empfiehlt sich die Lektüre von [5].

3.2.2 Typdefinitionen

XML Schema unterscheidet Typen nach verschiedenen, orthogonalen Kriterien. Einerseits existiert die Unterscheidung in einfache ("simple„) und komplexe ("complex„) Typen:

- Simple Typen sind atomar, dies bedeutet, dass der von ihnen definierte gültige Inhalt nicht in eine Zusammensetzung von Inhalten anderer Typen aufgeteilt werden kann. Daraus folgt, dass Elemente, die einen simplen Typ haben, keine Attributdefinitionen oder Definitionen für Kindelemente enthalten dürfen.
- Komplexe Typen sind nicht atomar und dürfen daher Attribute und Kinder definieren.

Weiterhin werden Typen nach Ihrer Herkunft unterschieden:

- Eingebaute Typen

  Stehen in allen XML Schema Definitionen zur Verfügung und sollten von jedem validierenden Parser unterstützt werden. Hier gibt es eine weitere Unterscheidung in primitive Typen, welche nicht auf anderen Typen aufbauen und solchen, welche dies tun -

den abgeleiteten Typen. Die für die Ableitung verwendeten Typen werden Basistypen genannt.
XML Schema verfügt über eine große Menge von bereits eingebauten Typen: So sind Typen zur Repräsentation von Zahlen in verschiedenen Variationen, verschiedene Arten von Zeichenketten, binären Daten und URIs enthalten. Weiterhin sind die von DTDs bekannten Typen ID und IDREF mit gleicher Funktionalität enthalten. Allerdings empfiehlt sich für Referenzen, bei Gebrauch von XML Schema, die Nutzung der „Ref“ und „Keyref“-Konstrukte, auf welche im Abschnitt „Weitergehende Definitionen“ eingegangen wird.

- Benutzerdefinierte Typen
  Diese werden vom Autor einer XML Schema Definition abgeleitet. Sie sind nur dort wo sie definiert wurden verfügbar. Diese Typen gehören auch zur Gruppe der abgeleiteten Typen.

Eine Unterscheidungsmerkmal von abgeleiteten Typen besteht in der Art wie sie ihre Basistypen verknüpfen: Es besteht die Möglichkeit die Wertebereiche mehrerer Basistypen zu vereinigen - was einem vereinigten Typ entspricht - oder aber eine Reihe von Inhalten eines Basistyps zu erlauben - was einem Listentypen entspricht.

Um benutzerdefinierte Typen in einer XML Schema Definition zu erstellen werden folgende syntaktische Elemente genutzt:

```
<xsd:simpleType name="typName">Inhalt</xsd:simpleType>
```

für die Erstellung eines simplen Typen und

```
<xsd:complexType name="typName">Inhalt</xsd:complexType>
```

für die Erstellung eines komplexen Typen.

Der Inhalt der Typendeklarationen kann aus verschiedenen weiteren Deklarationen bestehen. Beispielsweise können Elementdeklarationen, welche im Teil „Festlegen gültiger Elemente“ vorgestellt werden, und auch Attributdeklarationen, auf welche im Teil „Festlegen gültiger Element-Attribute“ genauer eingegangen wird, eingesetzt werden.

Bei der Deklaration mehrerer Elemente müssen diese gruppiert werden. Die kann durch folgende syntaktische Elemente geschehen:

- `<xsd:sequence>Elementdeklarationen</xsd:sequence>`
  Dies legt fest, dass die deklarierten Elemente in der definierten Reihenfolge vorkommen müssen.

- `<xsd:choice>Elementdeklarationen</xsd:choice>`

  Dies legt fest, dass genau eines der deklarierten Elemente vorkommen muss.

- `<xsd:all>Elementdeklarationen</xsd:all>`

  Dies legt fest, dass alle deklarierten Elemente genau einmal vorkommen müssen. Weiterhin dürfen hier deklarierte Elemente nur innerhalb dieser Gruppe vorkommen und nirgendwo sonst im Dokument.

Eine Deklaration eines komplexen Typen „adresse“ mit Hilfe einer Gruppierung könnte wie folgt aussehen:

```
<xsd:complexType name="adresse">
  <xsd:sequence>
    <xsd:element name="vorname" type="string"/>
    <xsd:element name="nachname" type="string"/>
    <xsd:element name="strasse" type="string"/>
    <xsd:element name="hausnummer" type="decimal"/>
    <xsd:element name="postleitzahl" type="decimal"/>
  </xsd:sequence>
</xsd:complexType>
```

Weiterhin gibt es auch Möglichkeiten um Attribute zu Gruppen zusammenzufassen auf welche hier nicht genauer eingegangen wird.

Um komplexe Typen zu erweitern wird das syntaktische Element „extension“ verwendet. Ein Beispiel hierfür:

```
<xsd:extension base="baseType">
  Element- und Attributdeklarationen
</xsd:extension>
```

Hiermit können zu einem Basistyp zusätzliche Attribute und Elemente hinzugefügt werden. Eine Erweiterung des oben beschriebenen Adresstyps um ein neues Element „postfach“ könnte wie folgt aussehen:

```
<xsd:complextType name="adresseMitPostfach">
  <xsd:complexContent>
    <xsd:extension base="adresse">
      <xsd:element name="postfach" type="numeric">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

In ähnlicher Weise lassen sich bestehende Typen auch einschränken.

Die Ableitung eines Listentypen von einem Basistyp geschieht durch den Einsatz von „<xsd:list>“. Das folgende Beispiel deklariert einen Typen „simpleList“, welcher eine leerzeichenseparierte Menge von Zahlen enthalten kann:

```
<xsd:simpleType name="simpleList">
  <xsd:list itemType="numeric"/>
</xsd:simpleType>
```

Zur Ableitung eines vereinigten Typen wird „<xsd:union>“ verwendet. Um beispielsweise einen Typ zu konstruieren, welcher entweder Zahlen oder einen boolschen Wert enthält würde man wie folgt vorgehen:

```
<xsd:simpleType name="simpleUnion">
  <xsd:union memberTypes="boolean number"/>
</xsd:simpleType>
```

Zusätzlich zu benannten Typendeklarationen besteht die Möglichkeit Typen direkt in einer Element- oder Attributdeklaration ohne Verwendung des name-Attributs zu deklarieren.

3.2.3 Festlegen gültiger Elemente

Die Festlegung eines gültigen Elemente kann wie folgt aussehen:

```
<xsd:element name="elementName">...</xsd:element>
```

Das Attribut „name“ legt dabei den Namen des beschriebenen Elements fest. Die Festlegung des gültigen Inhalts erfolgt über die Zuweisung eines Typs. Dies geschieht für benannte Typen durch Setzen des Attributs „type“ und für unbenannte Typen durch Deklaration des Typen im Inhalt des „<xsd:element>“-Elements. Die Anzahl der Vorkommen eines Elements in einem Dokument kann über die Attribute „minOccurs“ und „maxOccurs“ eingeschränkt werden. Werden diese beiden Attribute nicht angegeben so gilt für beide der Standardwert 1.

3.2.4 Festlegen gültiger Tag-Attribute

Die Definition gültiger Attribute erfolgt ähnlich wie die Definition von Elementen:

```
<xsd:attribute name="attributName" type="typ">
```

Da Attribute gleichen Namens nie mehrfach in einem Tag vorkommen können gibt es im Unterschied zum „<xsd:element>“-Tag nicht die Möglichkeit „minOccurs“ und „maxOccurs“ zu nutzen. Stattdessen wird das Attribut „use“ verwendet um anzuzeigen ob das spezifizierte Attribut vorkommen muss (use=“required”), ob es vorkommen darf (use=“optional”) oder ob es nicht vorkommen darf (use=“prohibited”). Weiterhin lassen sich Werte mit dem Attribut fixed=“Wert” festschreiben und mit default=“defaultWert” lassen sich Standardwerte festlegen, welche von validierenden Parsern eingefügt werden.

3.2.5 Weitergehende Definitionen

Die oben beschriebenen Elemente „<xsd:element>“ und „<xsd:attribute>“ lassen sich sowohl standalone direkt unterhalb des Wurzelements des XML Schema Dokuments einsetzen, als auch ineinander verschachteln. Um auf direkt unterhalb des Wurzelelement deklarierte Elemente und Attribute innerhalb der Definition eines Typs zugreifen zu können gibt es die Möglichkeit mit dem Attribut ref=“name” ein unter den Namen „name“ deklariertes Element bzw. Attribut zu referenzieren. Ein Beispiel hierfür:

```
<xsd:attribute name="adressat" type="string" />
<xsd:element name="brieftext" type="string" />

<xsd:complexType name="brief">
  <xsd:attribute ref="adressat" use="required"/>
  <xsd:element ref="brieftext">
</xsd:complexType>
```

Eines der Entwurfsziele von XML Schema war die Integration von XML mit relationalen Datenbanken zu erleichtern. Aus diesem Grund wurden aus diesem Bereich bekannte Konzepte übernommen:

- Referenzen

  Mit Hilfe der „<xsd:key>“ und „<xsd:keyref>“ Konstrukte können in XML Schema Primär-Fremdschlüssel ähnliche Beziehungen zwischen beliebigen Mengen von Elementen hergestellt werden. Dabei wird über „<xsd:key>“ festgelegt welche Elementmenge das referenzierte Objekt darstellt:

  ```
  <xsd:key name="nameofKey">
    <selector xpath="XPath Ausdruck1">
    <field xpath="XPath Ausdruck2">
  </xsd:key>
  ```

Hierbei legt „XPath Ausdruck1“ fest in welchem Scope die Schlüsseldeklaration gelten soll, in „XPath Ausdruck2“ wird die Elementmenge selbst ausgewählt. In einer Dokumentinstanz des XML Schemas muss die in „XPath Ausdruck2“ festgelegte Elementmenge im angegebenen Scope immer eindeutig sein. Referenzen auf einen mit „<xsd:key>“ deklarierten Schlüssel werden mit Hilfe von „<xsd:keyref>“ gelegt:

```
<xsd:keyref name="nameofKeyref">
  <selector xpath="XPath Ausdruck1">
  <field xpath="XPath Ausdruck2">
</xsd:keyref>
```

Hierbei legt „XPath Ausdruck1“ wieder den Scope der Referenz fest, während „XPath Ausdruck2“ die referenzierende Elementmenge beschreibt.

Diese Möglichkeit der Referenzierung hat gegenüber den aus DTD bekannten ID und IDREF-Attributen mehrere Vorteile: Zum einen wird nun die Möglichkeit einen Typ für das referenzierende und referenzierte Element zu setzen eröffnet, da die Referenzdeklaration von der Typdeklaration getrennt ist. Zum anderen können nicht mehr nur einzelne Attribute aufeinander verweisen sondern ganze Element- und Attributmengen auf andere Element- und Attributmengen. Weiterhin ist der Scope von Referenzen nicht mehr immer global sondern lässt sich sinnvoll einschränken. Hier kommt die Mächtigkeit von XPath zur Geltung.

- Abbildung von NULL-Werten

  Auch zur Darstellung von NULL-Werten ist in XML Schema eine Möglichkeit vorgesehen: So kann bei der Deklaration eines Elements mit Hilfe des Attributs „nillable“ angegeben werden dass dieses Element NULL sein kann. Um das Tag dann in einem Dokument tatsächlich auf NULL zu setzen wird das Attribut „xsi:null“ auf „true“ gesetzt. Beispiel: Im XML Schema Dokument:

  ```
  <xsd:element name="hugo" nillable="true"/>
  ```

  In der Dokumentinstanz:

  ```
  <hugo xsi:nill="true"/>
  ```

Ein weiterer Fortschritt von XML Schema gegenüber DTDs besteht darin dass sie selbstdokumentierend sein können: Durch Einsatz des „<xsd:annotation>“ Elements und seinem Kind „<xsd:documentation>“ kann Dokumentation über das Schema direkt in ihm selbst gepflegt

werden. Dies ist insbesondere dann wichtig, wenn XML Schema als Vertrag über die Struktur von ausgetauschten XML Daten betrachtet wird, da dann in den Annotationen weitere Informationen über das verwendete Schema dokumentiert werden können.

3.3 Grenzen von XML Schema

Da sich XML Schema seine Mächtigkeit mit einer hohen Komplexität erkauft bedingt dies, dass es wesentlich schwieriger zu erlernen ist, als die einfachereren DTDs. Weiterhin sind in XML Schema verfasste Strukturbeschreibungen meist länger als entsprechende DTDs und tendieren daher schneller dazu unübersichtlich zu werden. Weitere Kritikpunkte an XML Schema sind:

- Es ist nicht möglich in einem Schema den Namen des Wurzelelements für alle Dokumentinstanzen festzulegen.
- Bei der Beschreibung von gemischtem Inhalt, also Text und Elementen, können die Textdaten in keiner Weise eingeschränkt werden.
- Standardwerte für Elemente müssen immer Textdaten sein und können keine weiteren Elemente enthalten.
- Die Nutzung von XPath in den „key“ und „keyref“ Konstrukten wird von oft als unnötige Komplexität angesehen.

Insgesamt lässt sich jedoch sagen, dass XML Schema als Strukturbeschreibungssprache die graviernden Mängel von DTDs behebt und recht gut etabliert ist. Falls keine wichtigen Rahmenbedingungen dagegen sprechen, sollte XML Schema derzeit das Mittel der Wahl für Strukturbeschreibungen sein.

4 Weitere Strukturbeschreibungssprachen

Neben den beiden bereits vorgestellten Sprachen zur Strukturbeschreibung existieren zahlreiche andere Ansätze. Obwohl diese voraussichtlich nie die Bedeutung von DTDs und XML Schema erreichen werden, lohnt es sich dennoch einen kurzen Blick auf sie zu werfen, um die den jeweiligen Ansätzen zugrunde liegende Denkweise kennenzulernen.

4.1 Datatypes for DTDs

Das Ziel beim Sprachentwurf von Datatypes for DTDs (kurz: DT4DTD) war es die in den XML 1.0 [1] festgelegten DTDs um Datentypen zu erweitern. Dies eröffnet die Möglichkeit bestehenden Systemen, welche auf DTDs basieren und nicht komplett auf eine neue Technologie umgestellt werden sollen, dennoch den Gebrauch von Datentypen zu ermöglichen. Dabei wurden die Datentypen so gewählt, dass diese mit den in XML Schema zur Verfügung stehenden kompatibel sind, um eine spätere Migration auf XML Schema zu erleichtern. DT4DTD sind in einer W3C Note vom 13. Januar 2000 [11] beschrieben.

Der Syntax von DT4DTD ist dementsprechend eine Obermenge der DTD-Syntax. Es werden jedoch zwei neue Attribute eingeführt: “e-dtype” zur Beschreibung von Elementdatentypen und “a-dtype” zur Beschreibung von Attributdatentypen. Zur Verbindung der Namen von Datentypen mit ihrer Definition dienen spezielle NOTATION Deklarationen. Die Defintion des Datentypen selbst kann dann als XML Schema Datentypdefinition oder als in [11] vordefinierter XML Datentyp vorliegen. Ein Beispiel soll dies verdeutlichen:

```
<!NOTATION number
           SYSTEM "urn:schemas-microsoft-com:datatypes/number">

<!ELEMENT foo EMPTY>

<!ATTLIST foo
  a-dtype    CDATA   #FIXED
             "hausnummer number" >
```

Hier wird das Attribut „hausnummer“ des Elements „foo“ als Zahl vom Typ „number“ deklariert.

4.2 XML-Data Reduced

Die Sprache XML-Data Reduced (kurz: XDR) wurde dem W3C als Alternative zu XML Schema vorgelegt. Sie wurde von einem Firmenkonsortium entwickelt, an welchem auch Microsoft beteiligt war. XDR

ist vor allem deshalb interessant, weil es vollständig von Microsoft als Teil der .NET Initiative implementiert wurde. Weiterhin wird sie bereits von einem Browser - dem Internet Explorer 5 - und dem zu .NET gehörigen BizTalk Framework unterstützt.

Allerdings begann Microsoft nach der Verabschiedung des XML Schema Vorschlags als W3C Empfehlung mit der Umstellung ihrer Produkte auf XML Schema und bietet inzwischen Werkzeuge zur Migration von XDR nach XML Schema an.

XDR hat eine große Ähnlichkeit mit XML Schema: Grundsätzlich ist auch diese Sprache ein XML-Dialekt mit weitgehend äquivalenten Sprachkonstrukten. Beispielsweise erfolgt die Deklaration von Elementen und Attributen auf ähnliche Weise und auch viele der eingebauten Datentypen sind - bis auf ihren Namen - gleich. Allerdings ergeben sich grundlegende Unterschiede im Umgang mit Typen: Im Gegensatz zu XML Schema unterstützt XDR keine Erweiterung eingebauter Datentypen durch den Benutzer. Weiterhin werden Typen in XDR Element- bzw. Attributspezifisch deklariert, während in XML Schema der gleiche Typ sowohl für Elemente als auch für Attribute verwendet werden kann. Auch können Element- und Attributtypen in XDR nicht inline deklariert werden sondern sind auf separate Deklaration angewiesen. Als Beispiel hier der im Kapitel „XML Schema“ entworfene einfache Adresstyp in XDR:

```
<xdr:ElementType name="vorname" dt:type="dt:string">
<xdr:ElementType name="nachname" dt:type="dt:string">
<xdr:ElementType name="strasse" dt:type="dt:string">
<xdr:ElementType name="hausnummer" dt:type="dt:number">
<xdr:ElementType name="postleitzahl" dt:type="dt:number">

<xdr:ElementType name="adresse" order="seq">
    <xdr:element type="vorname"/>
    <xdr:element type="nachname"/>
    <xdr:element type="strasse"/>
    <xdr:element type="hausnummer"/>
    <xdr:element type="postleitzahl""/>
</xdr:ElementType>
```

Weitere Informationen zu XDR sind der XDR Schema Referenz [12] zu entnehmen.

4.3 The Schematron

Der meist nur als „Schematron“ bezeichnete Vorschlag „The Schematron“ repräsentiert einen, von allen bisher vorgestellten Sprachen grundverschiedenen, Ansatz zur Struktubeschreibung: Anstatt der Definition der Struktur mittels einer formalen Grammatik, wird hier versucht mit Hilfe von Mustern in der Baumdarstellung von XML zu arbeiten. Als Techniken zum Finden solche Muster wird XPath mit einer XSLT-Erweiterung verwendet. Aus diesem Grund funktioniert Schematron mit bereits existierenden XSLT-Implementationen und erfordert keine neuen Produkte. Es wurde so konstruiert, dass es als Ergänzung zu XML Schema eingesetzt werden kann.

Schematron macht sich zunutze, dass wohlgeformte XML Dokumente immer als Baum dargestellt werden können. Mittels XSLT Mustern können einzelne Teile dieses Baumes ausgewählt werden und Beziehung zu anderen Teilen gesetzt werden. Dies ermöglicht eine relativ einfache Definition von Regeln, welche sich mit Grammatikbasierten Sprachen nur schwer ausdrücken lassen. So lässt sich mittels Schematron leicht festlegen, dass ein Element immer ein bestimmtes anderes Element als Vater haben muss.

Den Kern von Schematron bilden Annahmen über die XML Daten, welche mittels eines „<assertion>“ Elements für positive Annahmen und mittels eines „<report>“ Elements für negative Annahmen ausgedrückt werden. Diese Annahmen können zu Regeln gruppiert werden, was mittels dem „<rule>“ Tag geschieht. Regeln wiederum können zu Mustern zusammengefasst werden. Die geschieht mit Hilfe der „<pattern>“ Elemente. Der Aufbau eines Schematron Dokumentes kann also strukturell wie folgt aussehen:

```
<schema>
  <pattern>
    <rule>
      <assert> oder <report>
        ...
      </assert> oder </report>
    </rule>
    ...
  </pattern>
  ...
</schema>
```

Annahmen können mittels eines Attributes „test“ festlegen was angenommen werden soll. Dabei bekommt „test“ ein XSLT-Pattern zugewiesen. Ist die Annahme wahr (bzw. falsch im Falle von „<report>“)

so wird der Inhalt des Elements ausgegeben um im Validationsbericht angezeigt zu werden.

Die Definition des Kontextes für welchen Annahmen gelten sollen, wird das „context“ Attribut des „<rule>“ Elements verwendet. Diesem kann ein XPath-Ausdruck zugewiesen werden. Die in der Regel eingeschlossenen Annahmen werden für den gesamten Kontext überprüft.

Zur Illustration folgt ein Beispiel für eines Schemtron Dokuments zur Validation des bereits bekannten Adresstypen. Zu beachten ist dass hier keine Typen validiert werden sondern vielmehr nur der korrekte Aufbau eines „<adresse>“ Elements.
Derzeit befindet sich Schematron in einem Standardisierungsprozess um in ISO 19757 als „Document Schema Definition Language“ aufgenommen zu werden.

```
<schema>
  <pattern name="Elemente">
    <rule context="vorname">
      <report test="child::*">
        <name/>
        Element sollte keine Kinder haben
      </report>
      <assert test="parent::adresse">
        <name/>
        Element muss Kind von Adresse sein
      </assert>
    </rule>
    <rule context="nachname">
      <report test="child::*">
        <name/>
        Element sollte keine Kinder haben
      </report>
      <assert test="parent::adresse">
        <name/>
        Element muss Kind von Adresse sein
      </assert>
    </rule>
    usw. für Strasse, Hausnummer und Postleitzahl ...
  </pattern>
</schema>
```

Beispiel 4.3.1

4.4 Schema for Object Oriented XML

Die Strukturbeschreibungssprache „Schema for Object Oriented XML“ (SOX) wurde von dem Unternehmen „Commerce One“ entwickelt und liegt als W3C Note [14] vor. Dabei wurde darauf geachtet aus der Objektorientierung bekannte Ansätze wie Polymorphie zu verfolgen. Damit ist SOX neben XDR eine weiter Alternative zu XML Schema mit ähnlicher Funktionalität: Sie unterstützt Erweiterung von Datentypen, Vererbung von Elementtypen sowie den Einsatz von Namespaces und Polymorphie. Ähnlich zu XML Schema bietet auch SOX die Möglichkeit Dokumentation in die Strukturbeschreibung zu integrieren.

5 Zusammenfassung und Ausblick

In dieser Ausarbeitung wurde die Rolle der Strukturbeschreibungssprachen im Kanon der XML-Familie erläutert: Ausgehend von einigen Beispielen für den Einsatz wurde die Notwendigkeit zur Strukturbeschreibung erarbeitet. Daran schloss sich eine Vorstellung der beiden wichtigsten Sprachen „Document Type Definitions“ und „XML Schema“ mit einigen ihrer syntaktischen Elemente an. Zum Abschluss wurden einige der Alternativen zu XML Schema vorgestellt, von welchen einige - zu den populären Konzepten - grundlegend verschiedene Ansätze verfolgen.

Mit der Standardisierung von XML Schema als W3C Note im Mai 2001 ist auf dem Gebiet der Strukturbeschreibungssprachen eine gewisse Ruhe eingekehrt. Vor der Standardisierung wurden zahlreiche Ansätze entwickelt und als Kandidaten eingebracht was einen Wildwuchs ans Tools und Implementationen verursachte. Mit XML Schema wurde einer der komplexesten und funktionsreichsten Vorschläge zum Standard erhoben, welcher zahlreiche verschiedene Anforderungen erfüllen kann. Gerade aus diesem Grund lohnt es sich sicherlich auch die alternativen Ansätze, welche teilweise aktiv weiterentwickelt werden, zu beobachten und in ausgewählten Situationen einzusetzen.

Literatur

[1] W3C Recommendation - Extensible Markup Language (XML) 1.0: http://www.w3.org/TR/REC-xml

[2] Liste von XML Schema Definitionen auf den Webseiten der Organization for the Advancement of Structured Information Systems: http://www.xml.org/xml/industry_industrysectors.jsp

[3] Vorstellung der DiML auf den Seiten der Humboldt-Universität Berlin: http://www.educat.hu-berlin.de/diss_online/dissinfo/texte_html/diml111.html

[4] Verschiedene Autoren: Professional XML 2nd Edition. Wrox Press, 2001

[5] Walmsley, Priscilla: Definitive XML Schema. Prentice Hall,2002

[6] W3C Recommendation - XML Schema Part 0: http://www.w3c.org/TR/xmlschema-0/

[7] W3C Recommendation - XML Schema Part 1: http://www.w3c.org/TR/xmlschema-1/

[8] W3C Recommendation - XML Schema Part 2: http://www.w3c.org/TR/xmlschema-2/

[9] W3C Recommendation - Document Object Model: http://www.w3.org/DOM/

[10] W3C Recommendation - Namespaces in XML: http://www.w3.org/TR/REC-xml-names/

[11] W3C Note - Datatypes for DTDs specification: http://www.w3c.org/TR/dt4dtd

[12] XDR Schema Reference: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/htm/xsd_xdrref_1pr9.asp

[13] Homepage des Schematron Projekts: http://xml.ascc.net/xml/resource/schematron/schematron.html

[14] W3C Note - Schema for Object-Oriented XML 2.0: http://www.w3.org/TR/NOTE-SOX/