1 Einleitung

Diese Seminararbeit zum Thema „Serviceorientierte Softwarearchitektur“ wird sich mit dem Thema „Überblick über XML-Technologien“ beschäftigen und bildet damit den Einstieg in die zugehörige Vortragsreihe.

Wie der Titel bereits andeutet, wird es um die Grundlagen der verschiedenen, auf XML basierenden, Technologien gehen, da der Umfang der Arbeit nicht ausreichen wird, um alle angesprochenen Teilaspekte von XML erschöpfend zu betrachten.

Die Gliederung dieser Arbeit wird sich an dem 2. Kapitel des Buches von Thomas Erl „Service-Oriented Architecture“ orientieren, das die Grundlage dieser Seminarreihe ist. Den Anfang wird dabei eine Übersicht über XML im Allgemeinen bilden. Danach folgen dann jeweils eine Übersicht über Konzepte und Syntax von DocumentType Definitions (DTD), XML Schema Definition Language (XSD), Extensible Stylesheet Language Transformations (XSLT), XML-Query Language (XQuery) sowie XML-Path Language (XPath).

Für das Verständnis dieser Arbeit wird eine rudimentäre HTML-Kenntnis vorausgesetzt, da die benutzten Grundbegriffe (Tag, Element, Dokument etc.) nicht weiter erläutert werden.

2 Extensible Markup Language (XML)

XML wurde vom World Wide Web Consortium (W3C) als eine erweiterbare Auszeichnungssprache entwickelt, die den plattformunabhängigen Austausch von Daten ermöglichen soll. Zu diesem Zweck wurde XML als eine Untermenge der weitaus komplexeren SGML (Standard Generalized Markup Language) entworfen, die sich gerade aufgrund ihrer Komplexität bei der breiten Mehrheit von Entwicklern nicht durchzusetzen vermochte.

Die XML-Spezifikation erhielt im Februar 1998 vom W3C den Status einer Empfehlung (→ Recommendation), was im Klartext heißt, dass Anbieter, die XML in ihren Produktion unterstützen wollen, sich an die vorgegebene Spezifikation halten müssen.

2.1 Konzept

XML-Dokumente ermöglichen einen plattformunabhängigen Datenaustausch zwischen verschiedenen Systemen. Dies wird zum einem dadurch ermöglicht, dass es sich bei diesen Dateien um reine Textdateien handelt und zum anderen der verwendete Zeichensatz im Kopf des Dokuments angegeben wird. Dadurch können verarbeitende Anwendungen ohne großen Aufwand auf den Inhalt der Dokumente zugreifen und auch einfache Lesbarkeit durch einen Menschen ist dadurch sichergestellt. Im Vergleich zu HTML (XML und HTML sind beide Nachfahren von SGML) wird in XML keinerlei Wert auf Darstellung der Daten gelegt. In einem Dokument befinden sich in der Reinform erstmal nur die Daten (ein XML-Stylesheet kann eingebettet werden, dazu aber später mehr) ohne Angaben darüber, wie diese einem (menschlichen) Betrachter dargestellt werden sollen. Dies ist aber auch nicht die „Kernkompetenz“ von XML.

Ein XML-Dokument besteht intern aus einer Baumstruktur von Elementen.

Wie jeder natürliche Baum auch, kann ein XML-Baum nur eine Wurzel haben, die sog. Dokument-Wurzel (→„document root“). Unter dieser Wurzel können Verarbeitungsanweisungen(optional), Kommentare (optional) und das Dokument-Element (→ „document-element“) hängen. Das Document-Element stellt die Wurzel aller Knoten im dar und darf pro Dokument nur einmal vorkommen. Ein Knoten mit dem gleichen Namen wie das Dokument-Element irgendwo im Baum ist ebenfalls verboten.

Unterhalb des Dokument-Elements können dann aber beliebige Knoten (in beliebiger Schachtelung) vorkommen. Sie müssen lediglich den Namenskonventionen (→„name characters“) für XML-Knoten entsprechen.

Die Daten in einem Dokument werden auch selbsterklärend genannt, weil sie durch die Namen der Tags eine Art Label erhalten. Dadurch ist auch hier die Lesbarkeit deutlich erhöht, weil z.B. nicht die Spalte abgezählt werden muss, um die Bedeutung des jeweiligen Datensatzes zu ermitteln. In einem Element mit dem Namen „Buchautor“ wird in den seltensten Fällen ein Preis stehen. Durch die Möglichkeit, die Tags frei zu benennen werden somit nicht nur die Daten selbst einfacher zu verstehen, sondern auch Programme, die diese lesen, da hier ebenfalls auf die Namen der Elemente zugegriffen wird.

2.2 Beispieldokument[1]

Anhand des folgenden Beispiel-XML-Dokuments werden im weitern Verlauf des Kapitels die einzelnen Merkmale von XML-Dokumenten erläutert. Weitere Beispiele wird es dann zu den einzelnen Gebieten nicht (oder nur sehr vereinzelt) geben. Die Ziffern rechts benennen jeweils einen Bereich des Dokuments, zu dem weiter unten Erklärungen folgen.

```
<?xml version=“1.0“ encoding=“iso-8859-1“ standalone=“no“?>                    1
<!--Kommentare und Verarbeitungsanweisungen sind hier erlaubt -->              2
<!DOCTYPE books SYSTEM “http://myserver.com/books.dtd”>                        3
<!-- weitere Kommentare und Verarbeitungsanweisungen erlaubt -->
<books>                                                                        4
  <book category=“reference“>                                                  5
    <author>Nigel Rees</author>                                                6
    <title>Sayings of the Century</title>
    <price>8.95</price>
  </book>
  <book category=”fiction”>                                                    7
    <author><![CDATA[Evelyn Waugh]]></author>
    <title>Sword of Honour</title>
    <price>12.99</price>
  </book>
</books>                                                                       8
<!--Kommentare und Verarbeitungsanweisungen sind hier erlaubt -->              9
```

[1] Vgl. XML Professional, S. 58

2.3 Dokumentstrukturen

Ein XML-Dokument wird in drei Bereiche aufgeteilt, wobei zwei davon optional sind. Den Anfang bildet der Prolog (1-3), gefolgt vom Rumpf mit den Daten (4-7) und zum Abschluss der Epilog.

2.3.1 Prolog (1-3)

Der Prolog bildet den (optionalen) Anfang eines XML-Dokuments. Er kann die XML-Deklaration (1), Kommentare (2), Processing-Instructions sowie die Deklaration des Dokumenttyps (3) enthalten.

2.3.1.1 Die XML-Deklaration (1)

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

Das Attribut **version** ist obligatorisch. Im Normalfall wird hier der Wert „1.0" stehen, da dieser der ersten Version des XML-Standards entspricht. Mittlerweile hat das W3C aber auch schon die Version 1.1 des XML-Standards veröffentlicht (Änderungen sind eher mit der Lupe zu suchen und damit zu vernachlässigen).

Das Attribut **encoding** gibt den benutzten Zeichensatz des Dokuments an. Um sicherzustellen, dass eine verarbeitende Anwendung den richtigen Zeichensatz zur Dekodierung der Daten eines Dokuments benutzt, sollte dieses Attribut gesetzt werden. Geschieht dies nicht, muss die Anwendung den verwendeten Zeichensatz selbst erkennen. Standardmäßig wird UTF-8 oder UTF-16 angenommen.

Mit dem Attribut **standalone** wird angegeben, ob dem Dokument eine externe Dokumenttyp-Definition (3) zugeordnet ist. Gültige Wert sind hier „yes" und „no".

2.3.1.2 Kommentare (2)

Ein Kommentar in XML beginnt und endet genauso, wie auch in HTML. Das Symbol für ein Kommentaranfang ist die Zeichenfolge „**<!--**" und für ein Kommentarende „**-->**". Innerhalb eines Kommentars dürfen keine zwei Bindestriche aufeinander folgen und er darf nicht innerhalb eines Tags definiert werden. Ansonsten ist innerhalb der Kommentar-Zeichen alles erlaubt, denn Entity-Referenzen werden nicht aufgelöst und Zeichenfolgen werden nicht geparst, so dass auch die sonst zu schützenden Markup-Zeichen vorkommen dürfen.

2.3.1.3 Dokumenttyp-Deklaration (3)

Mit der Dokumenttyp-Deklaration (Doctype Definition) wird einem validierenden Parser mitgeteilt, von welchem Typ das folgende Dokument ist. Eingeleitet wird die Deklaration durch die Zeichenfolge „<!DOCTYPE“. Im Anschluss daran folgt der Name des Dokumenttyps, der auch gleichzeitig der Name des Dokument-Elements sein muss. Das Schlüsselwort „SYSTEM“ gibt an, wie der Parser die (in diesem Fall externe) Definition finden kann, nämlich indem er der URL “http://myserver.com/books.dtd” folgt. Alternativ zu „SYSTEM“ kann auch „PUBLIC“ angegeben werden. Dieses Schlüsselwort wird gefolgt von einem URI und – falls der URI nicht aufgelöst werden kann – einer URL.
Anstatt auf eine externe Quelle zu verweisen, kann die Deklaration auch innerhalb des XML-Dokuments stattfinden.

Dann sieht die Deklaration wie folgt aus:

```
<![DOCTYPE books [
        <!ELEMENT book (author, title, price)>
        ...
]]>
```

Näheres zum Inhalt der Deklaration im Abschnitt →DTDs.

2.3.2 Rumpf (4-8)

Der Rumpf eines XML-Dokuments wird von dem Dokument-Element (4+8) festgelegt. Alles (einschließlich des Dokument-Elements) was sich innerhalb dieses Elements befindet, ist Bestandteil des Rumpfes. Der Rumpf enthält also die eigentlichen Daten eines Dokuments. Im Beispiel wird unter ein Element dargestellt, das zusätzlich ein Attribut (5) trägt und weitere Kindelemente hat. Das Element „author“ (6) ist ein einfaches Element, das lediglich einen Textknoten als Kind hat.
Beim zweiten „author“-Element (7) ist zusätzlich ein CDATA-Block definiert.

2.3.2.1 Attribute (5)

Attribute bestehen aus einem Name-Wert-Paar, wobei der Aufbau wie folgt ist:
`Attributname="Attributwert" oder Attributname='Attributwert'.`
Sowohl der Attributname als auch der Attributwert müssen den Bestimmungen zu Namen in XML entsprechen. Markup-Zeichen dürfen nicht in ihrer Literalform vorkommen, sondern müssen durch ihrer Entities ersetzt werden.

Innerhalb eines Start-Tags oder Leeres-Element-Tag darf ein Attribut nur einmal vorkommen.
Zusätzlich zu den frei definierbaren Attributen gibt es in XML einige Spezialattribute, deren Bedeutung von vornherein festgelegt ist. Diese Attribute sind:

- xml:lang → wird genutzt, um dem Parser Hinweise auf die Art der Codierung und vor allem der benutzten Sprache (z.B. en-US, de-DE, de-AT) des folgenden Inhalts zu geben.
- xml:space → um dem HTML-Attribute **<pre>** Rechnung zu tragen, gibt es das Attribut xml:space. Es gibt an, ob Whitespace innerhalb eines Textes durch einen Parser erhalten werden soll oder nicht. Bei Verwendung eines →validierenden Parsers darf das Attribut nur einen der folgenden Werte annehmen:
    - o „preserve“
    - o „default“

Das Verhalten eines Parsers, wenn er das Attribut xml:space antrifft, ist vom W3C nicht vorgeschrieben.

2.3.2.2 Elemente (6)

XML-Elemente bestehen aus einem öffnenden und schließenden bzw. dem Leeren-Element-Tag. Innerhalb eines Tags können beliebig viele Attribut-Wert-Paare untergebracht werden. Innerhalb eines Elements kann es weitere Elemente, einfachen Text und → CDATA-Blöcke geben.
Einfacher Text und Attributwerte dürfen die Zeichen „&“ und „<“ nicht ungeschützt enthalten, da diese zum Markup gehören. Sie müssen durch ihre zugehörigen Entities (→ „Entity“) „&“ bzw. „<“ ersetzt werden.
Der Name eines Tags muss mit einem Buchstaben, einem Unterstrich oder einem Doppelpunkt beginnen, wobei die Zeichenfolge „xml“ (in beliebiger Groß-/Kleinschreibung) verboten ist und kann dann aus beliebigen →name characters bestehen. Die Benutzung eines Doppelpunktes wird allerdings vom W3C nicht empfohlen, die dieser für die Abgrenzung von →Namespaces und → lokalem Elementnamen benutzt wird.

2.3.2.3 CDATA-Blöcke (7)

Diese Blöcke beginnen mit der Zeichenfolge „<![CDATA[" und enden mit „]]>“. Zwischen diesen beiden Symbolen sind alle Zeichen erlaubt, da der Text nicht

interpretiert/geparst wird. Die einzige Zeichenfolge, die nicht in einem CDATA-Block erlaubt ist, ist „]]>“, die Ende-Markierung eines solchen Blocks. Der Inhalt wird 1:1 ausgegeben.

2.3.3 Epilog (9)

Der Epilog wird von einem der Autoren der XML-Spezifikation als „real design error“ bezeichnet. Da es in XML kein eindeutiges Dokument-Ende-Zeichen gibt, lässt sich auch schwer bestimmen, wo der Dokument-Rumpf zu Ende ist und der Epilog beginnen soll. Daher wird von der Nutzung des Epilogs dringend abgeraten, weil es bestenfalls zu zufälligem Verhalten führt.

3 DocumentType Definition (DTD)

3.1 Konzept[2]

DTDs dienen der Vereinheitlichung und Verdeutlichung von Dokumentenstrukturen. Mit ihrer Hilfe ist es möglich, wohlgeformte Dokumente auch auf ihre (syntaktische und semantische) Gültigkeit hin zu überprüfen.

Die Tatsache, dass ein XML-Dokument →wohlgeformt ist, hilft einem Anwendungsprogrammierer noch nicht wirklich weiter. Erhält er aus einer fremden (heißt entfernten) Quelle ein XML-Dokument, das seine Anwendung verarbeiten soll, müsste er zuerst die formale Korrektheit des Dokuments überprüfen, bevor die eigentliche Datenverarbeitung beginnen kann. Ansonsten bestünde die Gefahr, dass die eigene (oder abhängige) Anwendungen Fehler –

oder im schlimmsten Fall sogar Abstürze – produzieren, nur weil das gelieferte Dokument nicht den vereinbarten Spezifikationen entsprach.

Mit DTDs wird die Möglichkeit geboten, eben diese Gefahren zu eliminieren bzw. den Validierungsaufwand deutlich zu reduzieren. Die Kombination aus einem validierenden Parser und einer DTD würde dann nämlich die vollständige Arbeit übernehmen. Das einzige, was der verarbeitenden Anwendung dann noch zu tun übrig bliebe, wäre, einen definierten Fehlerzustand zu erzeugen oder im Normalfall, die gelieferten Daten zu verarbeiten.

Eine DTD hat auch noch weiteren Nutzen. Sie stellt implizit eine Dokumentation dar, die von anderen Entwicklern gelesen und berücksichtigt werden kann. Es ist einfacher anhand einer DTD die möglichen Strukturen eines XML-Dokuments zu ermitteln, als z.B. anhand einer Menge von einzelnen Dokumenten, die im groben zwar identisch sind, sich aber dennoch im Detail unterscheiden.

Ebenso wird die eigentliche Problemstellung, zu deren Lösung die gelieferten XML-Dokumente beitragen, durch eine DTD zentral festgelegt.

3.2 Syntax

Die Syntax der DocumentType Definitions (DTDs) unterscheidet sich von der allgemeinen XML-Syntax. Das hat auch dazu geführt, dass die Akzeptanz von DTDs lange Zeit nicht sonderlich hoch war. Außerdem wartet die Syntax mit Schwierigkeiten auf, die es nicht erlauben, komplexe Zusammenhänge zu definieren oder gar Datentypen von Attributen oder Knoten“texten“ festzulegen.

[2] Vgl. XML Professional, S. 89f

Nichtsdestotrotz sind DTDs sehr gut geeignet, um XML-Strukturen zu definieren und damit zur einheitlichen Handhabung der Dokumente beizutragen.
Im folgenden Kapitel → *XML Schema Definition Language* wird dann eine mächtigere Variante vorgestellt, wie man XML-Strukturen im Vorfeld definieren kann.

3.2.1 Elemente

Ein XML-Dokument besteht in den meisten Fällen ausschließlich aus Knoten und Attributen. Daher hier die Syntax für die Definition eines Elements in einer DTD.

Deklaration	**Bedeutung**
<!ELEMENT foo EMPTY>	Ein leeres Element mit dem Namen „foo“ (<foo/>)
<!ELEMENT foo ANY>	Ein Element „foo“ mit beliebigem Inhalt. Es können beliebige Knoten und Texte folgen.
<!ELEMENT foo #PCDATA>	Ein Element „foo“ mit einem Text als Kind (<foo>bar</foo>)
<!ELEMENT foo (A, B)>	Ein Element „foo“ mit je einem Kind A und B
<!ELEMENT foo (A | B)>	Ein Element „foo“ mit entweder einem Kind A oder B
<!ELEMENT foo (A?, B)>	Ein Element „foo“ mit einem optionalen Kind A und einem Kind B
<!ELEMENT foo (A*, B)>	Ein Element „foo“ mit beliebig vielen (auch 0) Kindern A und genau einem Kind B
<!ELEMENT foo (A, B)*>	Ein Element „foo“ mit beliebig vielen Sequenzen von Kindern A und B

Alle Elemente (A-E) müssen natürlich ebenfalls in der DTD deklariert werden.

3.2.2 Operatoren

Operator	**Bedeutung**
,	Sequenz
|	Auswahl
?	Optionalität eines Elements
*****	Beliebig häufiges Auftreten (auch kein Mal)
+	Mindestens ein Vorkommen

3.2.3 Attribute

Die grundlegende Syntax für eine Attributdeklaration ist Folgende:

```
<!ATTLIST elementName attributName1 AttributTyp Vorgabewert
                      attributName2 AttributTyp Vorgabewert>
```

Zu den einzelnen Bestandteilen dieser Deklaration:

Stichwort	**Bedeutung**
elementName	Knotenname, an dem das Attribut hängt
attributName	Name des Attributs
→ AttributTyp	ID, IDREF, IDREFS, CDATA, ENTITY, ENTITIES, NMTOKEN, NMTOKENS, NOTATION, Wert aus einer Aufzählung
→ Vorgabewert	#REQUIRED, #IMPLIED, #FIXED plus Vorgabewert, Vorgabewert

3.2.4 Attributtypen:

Attributtyp	**Bedeutung**
ID	Stellt einen eindeutigen Schlüssel im Dokument dar. Muss den Bedingungen zu Namen in XML gehorchen und darf nur einmal benutzt werden. Der → Vorgabewert muss #REQUIRED oder #IMPLIED sein.
IDREF	Definiert eine Referenz auf einen eindeutigen Schlüssel (→ ID). Damit kann einen n:1-Beziehung abgebildet werden, z.B. für den Austausch von Daten mit relationalen Datenbanken. Der Wert des Attributs muss eine vorhandene ID sein.
IDREFS	Wie IDREF, nur dass hier eine Menge von Werten mit Leerzeichen getrennt angegeben werden kann.
CDATA	Beliebiger Text ohne Markup-Symbole.
→ ENTITY	Name eines vorher definierten Entitys.
ENTITIES	Entsprechend IDREFS eine Menge von ENTITYs
NMTOKEN	NMTOKEN = name token. Legt ein Symbol innerhalb eines Dokuments fest. Die Werte für NMTOKENs unterliegen grundsätzlich den gleichen Beschränkungen wie Element- und Attributnamen, dürfen zusätzlich aber auch mit Ziffern beginnen. Eine Häufigkeitsbeschränkung existiert nicht. Name tokens sind eine Art offene Liste für ein Attribut. Die Gültigkeitsprüfung wird damit vom

	Parser an die rufende Anwendung abgewälzt (im Gegensatz zur Aufzählung)
NMTOKENS	Entsprechend IDREFS eine Menge von NMTOKENs.
NOTATION	Eine NOTATION ist eine Verknüpfung mit nicht-XML-Daten. So kann z.B. ein Bild mit einem bestimmten Element über eine Notation verbunden werden.
Wert aus einer Aufzählung	Hier kann eine benutzerdefinierte Liste von zulässigen Werten für ein Attribut angegeben werden. Die Liste ist folgendermaßen aufgebaut: (Wert1 | Wert2 | … | Wert n)

3.2.5 Entity

Entities in einem XML-Dokument können als Platzhalter für sich wiederholende Inhalte betrachtet werden. Man unterscheidet hierbei zwischen General- und Parameter-Entities und außerdem zwischen geparsten- und ungeparsten (parsed entities) Entities.

Der Unterschied zwischen geparsten- und ungeparsten Entities liegt – wie der Name schon vermuten lässt – darin, wie sie vom XML-Parser verarbeitet werden. Die geparsten Entities werden von einem →validierenden Parser erweitert und ausgewertet. Nach der Erweiterung muss das XML-Dokument immer noch wohlgeformt sein.

Die Erzeugung von (direkten und indirekten) Zirkelbezügen zwischen Entities ist verboten.

3.2.5.1 General Entities

Dies sind einfach nur Textbausteine, die an den entsprechenden Stellen im XML-Dokument eingefügt werden können. Es gibt keine Beschränkung darüber, wie oft ein Entity eingesetzt werden darf.

```
<!ENTITY copyright "Dieses Dokument &copy; 2005 durch den Autor">
```

Der Aufruf des so definierten Entities erfolgt dann einfach über „**&copyright;".**

<element>&copyright;</element> → <element> Dieses Dokument © 2005 durch den Autor</element>

Anstatt den Inhalt eines Entities direkt in der DTD abzulegen, kann man auch auf externe Quellen verweisen. Dafür wird eine Notation wie bei der DOCTYPE Definition benutzt:

```
<!ENTITY bar SYSTEM "http://www.someServer.com/myEntity.txt">
```

oder:

```
<!ENTITY bar PUBLIC "someMachine/Entities"
"http://www.someServer.com/myEntity.txt">
```

3.2.5.2 Parameter Entites[3]

Geparste Entities, die nur innerhalb einer DTD auftauchen, werden Paramter-Entities genannt. Mit ihrer Hilfe ist es möglich, häufige Definitionen selbst in einem Entity zu definieren und dann an den betreffenden Stellen in der DTD nur die Referenz auf das Entity einzufügen. Die Grundregeln für das Markup-Escaping müssen hierbei natürlich beachtet werden (d.h. nach der Expansion durch den Parser muss weiterhin wohlgeformtes XML vorliegen).

Ein Parameter-Entity wird folgendermaßen deklariert:

```
<!ENTITY % foo “Alter CDATA #IMPLIED Gewicht CDATA #IMPLIED Groesse CDATA #REQUIRED“>
```

Dieses Entity kann jetzt innerhalb einer zu erzeugenden Attributliste (ATTLIST) aufgerufen werden. Damit würde man sich die Arbeit ersparen, die darin definierten Attribute „Alter“, „Gewicht“ und „Groesse“ manuell einzufügen. Dies macht natürlich nur dann Sinn, wenn diese Attribute auch noch an anderen Stellen innerhalb der DTD benötigt werden.

```
<ATTLIST VersichertePerson
        %foo;
        Beruf CDATA #REQUIRED>
```

Das Element „VersicherstePerson“ verfügt jetzt über vier Attribute, nämlich die drei aus dem Entity, sowie ein explizit definiertes Attribut „Beruf“.

Im Gegensatz zu normalen Entities werden Parameter-Entities nicht mit dem Ampersand, sondern mit dem Prozentzeichen eingeleitet.

[3] Beispiel: vgl. XML Professional, S. 97

4 XML Schema Definition Language (XSD)

4.1 Konzept

„Der Zweck eines XML-Schemas ist es, eine Klasse von Dokumenten zu definieren […]“[4]. Ähnlich wie bei den →DTDs können Elemente und deren Aufbau sowie deren Attribute definiert werden. XML-Schemas bieten darüber hinaus noch viele weitere Funktionen, die man bei den DTDs vermisst hat. So ist die XSD-Language (zur Vereinfachung wird nur noch XSD oder Schema/s verwendet) vollständig getypt, d.h. es bestehen viele vorgefertigte Datentypen, wie man sie auch aus „normalen“ Programmiersprachen kennt und man hat die Möglichkeit, eigene Datentypen (auch durch Vererbung) zu erstellen. Zu diesem Zweck existieren in XSD die sog. „simpleTypes“ und „complexTypes“. Dazu später aber mehr. Ein weiterer Vorteil von Schemas gegenüber von DTDs ist die Tatsache, dass diese in der XML-Syntax verfasst werden und somit auch von einfachen XML-Parsern interpretiert werden können.

Insgesamt ist XSD eine sehr mächtige Möglichkeit, die Integrität von Dokumenten sicherzustellen und erleichtert die Anwendungsprogrammierung ungemein, da die Gültigkeitsprüfung jetzt vom →validierenden XML-Parser übernommen wird und nicht mehr in der Anwendung stattfinden muss, wie es bei DTDs der Fall war (auch hier hat zwar eine rudimentäre Gültigkeitsprüfung stattgefunden, aber wenn an einem Knoten z.B. nur Zahlwerte erlaubt sein sollen, muss dies explizit geprüft werden, weil es als Datentyp nur CDATA gibt).

4.2 Syntax

Da es in XML-Schema zu viele eingebaute Datentypen und Elemente gibt, wird an dieser Stelle lediglich anhand einiger Beispiele ein Überblick über die wichtigsten Bestandteile geliefert.

Die vollständige Empfehlung des W3C mit ausgiebigen Erklärungen können unter http://www.w3.org/XML/Schema#dev eingesehen werden.

[4] Quelle: http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/, 2005-11-10

4.2.1 Schema-Deklaration

Am Anfang eines jeden Schemas muss ein <schema>-Element mit dem Namensraum für XML-Schema stehen. Im allgemeinen sieht diese Definition dann so aus:

```
<xsd:schema xmlns:xsd="http://w3.org/2001/XMLSchema">
```

Die Namensraumabkürzung „xsd" für Schema wird nur laut Konvention benutzt und kann auch beliebig geändert werden. Wichtig ist nur, dass alle eingebauten Schema-Typen mit dem jeweils verwendeten Namensraum versehen werden. Im Folgenden wird immer das Kürzel „xsd" benutzt.

4.2.2 Elemente

Die Deklaration eines Elements erfolgt über

```
<xsd:element name="myElement" type="xsd:string">
```

In diesem Beispiel wurde ein Element mit dem Namen "myElement" deklariert. Als einziger Inhalt ist ein Text erlaubt. Für das type-Attribut können sämtliche →simpleTypes (vordefiniert und eigene) und →complexTypes angegeben werden.

Elementname	Kurzbeschreibung
all	Fasst Elemente zusammen, die entweder alle einmal oder keinmal in beliebiger Reihenfolge vorkommen dürfen
annotation	Möglichkeit, Dokumentation (für Mensch oder Maschine) festzulegen
any	Erlaubt beliebigen, wohlgeformten XML-Inhalt
anyAttribute	Wie "any", erlaubt es beliebige Attribute
appinfo	Kind von "annotation", dient der Maschinendokumentation
attribute	Ein Attribut als Kind eines Elements
attributeGroup	Eine benamte Gruppe von Attributen. Kann aus einer complexType-Deklaration (n-mal) referenziert werden.
choice	Erlaubt eine optionale Gestaltung des Instanzdokuments. Jeder Knoten darf genau eine Ausprägung der Kinder von choice haben.
complexContent	Legt den Inhalt eines Knotens derart fest, dass kein „Text" erlaubt ist. Nur Elemente und Attribute dürfen unterhalb des Knotens erscheinen.
complexType	Einleitung einer Typdefinition.
documentation	Kind von „annotation", dient der Dokumentation für menschliche Leser.
element	Ein XML-Knoten.
enumeration	Ein Aufzählungselement. Der Zielknoten darf dann nur Werte enthalten, die in den enumeration-Werten vorgegeben sind.
extension	Dient der Erweiterung eines simpleTypes zu einem complexType.

field	Legt das Element oder Attribut fest, dass in einem bestimmten Bereich eindeutig sein muss.
group	Eine benamte Gruppe von Elementen. Kann aus einer complexType-Deklaration (n-mal) referenziert werden.
import	Übername eines Elements oder Typs aus einem anderen Namensraum.
include	Einbindung eines anderen Schema(-Dokuments)
key	Legt einen Schlüssel fest, der von außen referenziert warden kann
keyref	Legt eine Referenz auf einen vorher definierten “key” fest
length	Legt die Listenlänge einer Liste eines vorher definierten Listentyps fest.
list	Definiert eine Liste von Elementen
maxInclusive	Legt den maximal zulässigen Wert eines Elementinhalts fest (bei simpleType Zahl- und Datumstypen)
maxLength	Legt die maximale Länge einer Liste eines vorher definierten Listentyps fest.
minInclusive	Legt den minimal zulässigen Wert eines Elementinhalts fest (bei simpleType Zahl- und Datumstypen)
minLength	Legt die minimale Länge einer Liste eines vorher definierten Listentyps fest.
pattern	Legt ein Muster (regulärer Ausdruck) fest, dem ein Elementinhalt vom Typ String (oder eines geerbten Typs) entsprechen muss
redefine	Ermöglicht es, Typen, Gruppen und Attribute aus externen Schemas umzudefinieren.
restriction	Leitet Beschränkungen geerbter Datentypen bei der Deklaration eines eigenen Typs fest.
schema	Leitet das Schema ein und enthält außerdem die Namenräume.
selector	Legt den Bereich fest, indem ein bestimmtest Element oder Attribut eindeutig sein muss.
sequence	Legt eine genaue Reihenfolge von Elementen, Gruppen und Attributen fest.
simpleContent	Leitet einen “simpleContent” (keine Elemente oder Attribute) ein
simpleType	Einleitung einer Typdefinition.
union	Legt fest, dass (und welche) ein Element Instanzen verschiedener Typen enthalten darf.
unique	Einleitung einer Eindeutigkeitsdefinition

Tab x.1: Elemente in XML-Schema[5]

[5] Quelle: http://www.w3.org/TR/xmlschema-0/#indexEl

4.2.3 Typedeklarationen

4.2.3.1 simpleType[6]

Die sog. simpleTypes beinhalten alle eingebauten Datentypen sowie benutzerdefinierte Datentypen, die von den eingebauten Typen abgeleitet sind. Kindelemente und Attribute sind in diesen Typen nicht erlaubt.

```
<xsd:simpleType name="myInteger">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="10000"/>
    <xsd:maxInclusive value="99999"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>
```

Das erste Beispiel deklariert einen Typ „myInteger“ der vom eingebauten Typen „xsd:integer“ abgeleitet ist und begrenzt die zulässigen Werte auf 10000 bis 99999 (jeweils inklusive).
Das Element <xsd:restriction> gibt an, welcher eingebaute Datentyp die Grundlage für den eigenen Typen ist und welcher Werte hier erlaubt sein sollen.
Im zweiten Beispiel wird auf der Basis des String-Datentyps der Datentyp „SKU“ definiert. Als Wertebereich wird hier mit Hilfe der eingebauten regulären Ausdrücke das Format des Textes derart festgelegt, dass er aus drei Ziffern gefolgt von einem Bindestrich und zwei Großbuchstaben aufgebaut sein muss.

4.2.3.2 complexType[7]

Ein complexType in XML-Schema ist ein zusammengesetzter Datentyp, der im Allgemeinen aus einer Folge von Elementen, Attributen und Referenzen besteht.

```
<xsd:complexType name="USAddress">
  <xsd:sequence>
    <xsd:element name="name"   type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="city"   type="xsd:string"/>
    <xsd:element name="state"  type="xsd:string"/>
    <xsd:element name="zip"    type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```

[6] Beispiel entnommen aus: http://www.w3.org/TR/xmlschema-0/#ref7

[7] Beispiel entnommen aus: http://www.w3.org/TR/xmlschema-0/#ref2

Das Beispiel zeigt einen Typ „USAddress“, der eine Folge von fünf Elementen und einem Attribut ist. Die Knoten sind jeweils wieder von einem bestimmten Typ (in diesem Fall eingebaute simpleTypes). Das Attribut ist vom Typ NMTOKEN, der bereits aus den DocumentType Definitions bekannt ist.

Ein gültiger Instanzknoten könnte dann folgendermaßen aufgebaut sein:

```
<billTo country="US">
  <name>Robert Smith</name>
  <street>8 Oak Avenue</street>
  <city>Old Town</city>
  <state>PA</state>
  <zip>95819</zip>
</billTo>
```

4.3 Beispiel Schema (Zusammenfassung)[8]

Zum Abschluss des Kapitels noch ein vollständiges XML-Schema, in dem die vorher als Beispiel deklarierten Bestandteile zusammen aufgeführt sind.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:annotation>
    <xsd:documentation xml:lang="en">
     Purchase order schema for Example.com.
     Copyright 2000 Example.com. All rights reserved.
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

  <xsd:element name="comment" type="xsd:string"/>

  <xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
      <xsd:element name="shipTo" type="USAddress"/>
      <xsd:element name="billTo" type="USAddress"/>
      <xsd:element ref="comment" minOccurs="0"/>
      <xsd:element name="items"  type="Items"/>
    </xsd:sequence>
    <xsd:attribute name="orderDate" type="xsd:date"/>
  </xsd:complexType>

  <xsd:complexType name="USAddress">
    <xsd:sequence>
      <xsd:element name="name"   type="xsd:string"/>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city"   type="xsd:string"/>
      <xsd:element name="state"  type="xsd:string"/>
      <xsd:element name="zip"    type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="country" type="xsd:NMTOKEN"
                   fixed="US"/>
  </xsd:complexType>
```

[8] http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/

```
  <xsd:complexType name="Items">
    <xsd:sequence>
      <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="productName" type="xsd:string"/>
            <xsd:element name="quantity">
              <xsd:simpleType>
                <xsd:restriction base="xsd:positiveInteger">
                  <xsd:maxExclusive value="100"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:element>
            <xsd:element name="USPrice"  type="xsd:decimal"/>
            <xsd:element ref="comment"   minOccurs="0"/>
            <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
          </xsd:sequence>
          <xsd:attribute name="partNum" type="SKU" use="required"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <!-- Stock Keeping Unit, a code for identifying products -->
  <xsd:simpleType name="SKU">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-[A-Z]{2}"/>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

4.4 Beispiel-XML-Dokument[9]

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
   <shipTo country="US">
      <name>Alice Smith</name>
      <street>123 Maple Street</street>
      <city>Mill Valley</city>
      <state>CA</state>
      <zip>90952</zip>
   </shipTo>
   <billTo country="US">
      <name>Robert Smith</name>
      <street>8 Oak Avenue</street>
      <city>Old Town</city>
      <state>PA</state>
      <zip>95819</zip>
   </billTo>
   <comment>Hurry, my lawn is going wild<!/comment>
   <items>
      <item partNum="872-AA">
         <productName>Lawnmower</productName>
         <quantity>1</quantity>
         <USPrice>148.95</USPrice>
```

[9] http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/

```
         <comment>Confirm this is electric</comment>
      </item>
      <item partNum="926-AA">
         <productName>Baby Monitor</productName>
         <quantity>1</quantity>
         <USPrice>39.98</USPrice>
         <shipDate>1999-05-21</shipDate>
      </item>
   </items>
</purchaseOrder>
```

5 Extensible Stylesheet Language Transformations (XSLT)

5.1 Konzept

Bei den Extensible Stylesheet Language Transformations werden XML-Dokumente mit Hilfe eines XML-Stylesheets (XSL) in ein anderes XML-Dokument umgewandelt. Dabei wird die Struktur des Eingangsdokuments dahingehend bearbeitet, dass es der – im XSL festgelegten – Ausgangsstruktur entspricht.

Der dabei eingesetzte XSL-Prozessor, ohne den keine Transformation stattfinden kann, arbeitet dabei ausschließlich auf der Struktur der Dokumente und nicht auf den darin stehenden Daten. Diese Informationen beschafft sich der Prozessor über APIs z.B. aus der DOM-Darstellung (Document Object Model) des Eingangs-Dokuments.

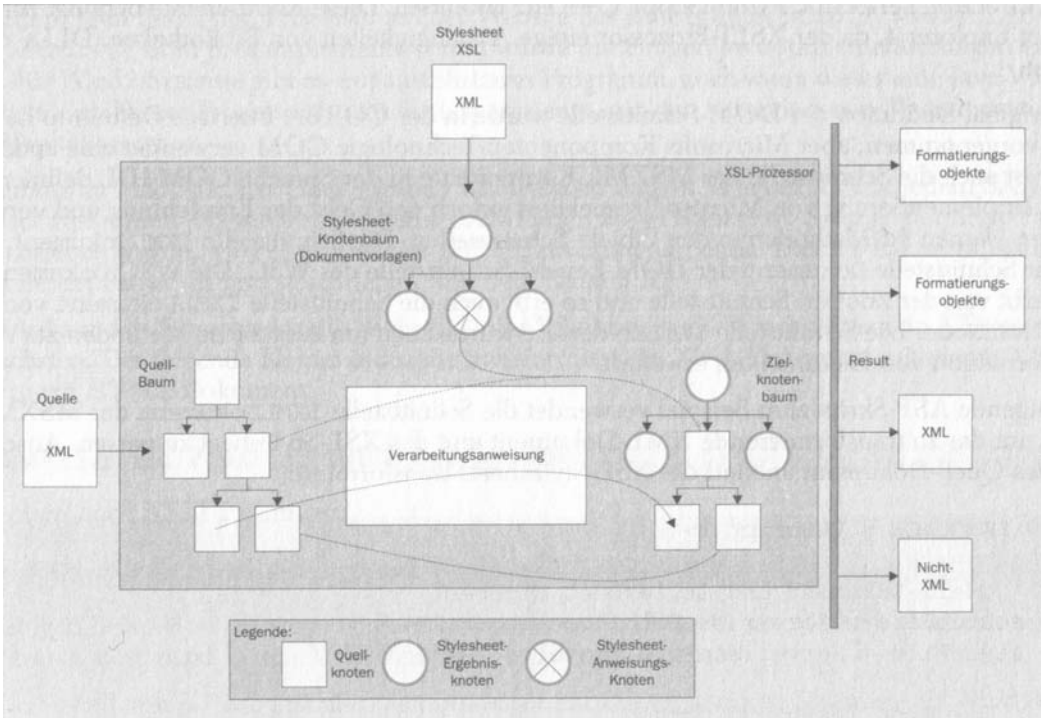

Abb. 5.1: Arbeitsweise eines XSL-Prozessors[10]

Um die Knoten eines Eingangsdokuments zu selektieren, wird mit →XPath-Ausdrücken gearbeitet.

[10] Quelle: XML Professional, S. 377

5.2 Exkursion XML-APIs

Derzeit existieren zwei große APIs für XML. Zum einen ist das das „Doucment Object Model“ (DOM) und die „Simple API for XML“ (SAX). Das DOM zeichnet sich dadurch aus, dass ein XML-Baum komplett in den Speicher gelesen wird und damit Navigation vorwärts und rückwärts durch ein Dokument kein Problem sind. Allerdings kann es bei großen Dokumenten leicht zu Performance-Problemen kommen.

SAX auf der anderen Seite durchläuft den Baum zur Laufzeit und hält immer nur einzelne Fragmente des Baums im Speicher. Für jeden Knoten wird dann ein Ereignis ausgelöst und eine entsprechende Verarbeitung initiiert. Bei großen Dokumenten ist diese Vorgehensweise performanter.

Die beiden APIs sind also vollständig komplementär und haben beide ihre jeweiligen Anwendungsgebiete.

5.3 Syntax

Da es sich bei XSL-Dokumenten grundsätzlich um wohlgeformte XML-Dokumente handelt, ist die grundlegende Syntax identisch.

Für XSL-Stylesheets gibt es einen speziellen Namensraum, in dem die XSL-Elemente definiert sind. Durch Angabe des Namesraums „http://www.w3.org/1999/XSL/Transform“ im einleitenden <stylesheet> -Tag, wird dem Parser mitgeteilt, dass für die folgende Verarbeitung ein XSL-Prozessor benötigt wird, um die Transformation durchzuführen.

Stylesheets können sowohl als externe Dokumente, aber auch in ein XML-Dokument eingebettet auftreten.

Im Folgenden wird eine kurze Übersicht über die im o. g. Namensraum vorhandenen Elemente gegeben. Details zur Empfehlung des W3C sind unter http://www.w3.org/TR/xslt bzw. http://www.w3.org/TR/xslt#element-syntax-summary verfügbar.

Element	**Beschreibung**
apply-imports	Ruft importierte Templates auf, die durch das aktuelle Dokument überlagert wurden
apply-templates	Sucht nach passenden Templates innerhalb des Stylesheets
attribute	Erstellt ein Attribut
attribute-set	Erstellt eine benamte Menge von Attributen, die später eingebunden werden kann.
call-template	Ruft ein Template mit Namen auf.

choose	Stellt eine Verarbeitungsauswahl dar.
comment	Fügt einen Kommentar in die Ausgabe ein.
copy	Erstellt eine Kopie des aktuellen Knotens in der Ausgabe. Kinder und Attribute werden nicht mitkopiert.
copy-of	Erstellt eine Kopie eines Baumfragments in der Ausgabe.
decimal-format	Legt eine benamte Formatierung für Zahlwerte fest.
element	Erstellt ein Element in der Ausgabe.
fallback	Bietet einen Rücksprung zu einer früheren Version, falls ein Parser ein Element oder eine Anweisung nicht unterstützt.
for-each	Eine Iteration über alle Elemente eines bestimmten Musters.
if	Eine bedingte Verarbeitung.
import	Importiert ein externes XSL in das aktuelle Stylesheet. Dabei wird ein Importbaum erstellt, der die Reihenfolge der Abarbeitung der Templates beeinflusst.
include	Fügt an der Stelle des include-Elements den gesamten Inhalt der eingefügten externen Quelle ein, der zwischen den <stylesheet>-Tags steht.
key	Definiert einen Index.
message	Sendet eine Textnachricht an den Ausgabepuffer oder einen Dilogbox. Außerdem wird ein Systemfehler geworfen.
namespace-alias	Wechselt ein Namensraum-Präfix durch ein anderes aus.
number	Fügt eine formatierte Zahl in die Ausgabe ein.
otherwise	Default-Fall in einem →choose
output	Gibt das Ausgabeformat an (HTML, XML,…)
param	Ein benamter Parameter im Stylesheet oder Template
processing-instruction	Erzeugt eine Verarbeitungsanweisung in der Ausgabe
sort	Ermöglicht eine sortierte Verarbeitung bei einem →for each oder →apply-templates
strip-space	Legt fest, welche nur-Whitespace-Knoten innerhalb eines →elements entfernt werden sollen
stylesheet	Anfang eines Stylesheets. Hier warden auch die verwendeten Namensräume angegeben.
template	Anfang eines Templates, nachdem Elemente verarbeitet werden sollen.
text	Erzeugt Text in der Ausgabe und bietet Kontrolle über die Behandlung von Whitespace.
transform	Synonym für →stylesheet
value-of	Gibt den Wert eines Knotens oder Attributs aus.
variable	Definiert eine Variable, auf die spatter zugegriffen werden kann. Variablen können nicht mehrmals definiert werden (Ausnahme in einem →for each)

when	Eine Bedingung im →choose
with-param	Parameterübergabe beim Aufruf von →apply-templates und →call-template

5.4 Anwendungsbeispiel für XSL-Transformationen[11]:

5.4.1 XML-Dokument:

```
<doc>
  <title>Document Title</title>
  <chapter>
    <title>Chapter Title</title>
    <section>
      <title>Section Title</title>
      <para>This is a test.</para>
      <note>This is a note.</note>
    </section>
    <section>
      <title>Another Section Title</title>
      <para>This is <emph>another</emph> test.</para>
      <note>This is another note.</note>
    </section>
  </chapter>
</doc>
```

5.4.2 Stylesheet:

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"          1
                xmlns="http://www.w3.org/TR/xhtml1/strict">
<xsl:strip-space elements="doc chapter section"/>                         2

<xsl:output method="xml" indent="yes" encoding="iso-8859-1"/>             3

<xsl:template match="doc">
 <html>
   <head>
     <title>                                                              4
       <xsl:value-of select="title"/>
     </title>
   </head>
   <body>
     <xsl:apply-templates/>                                               5
   </body>
</html>
</xsl:template>

<xsl:template match="doc/title">
  <h1>
    <xsl:apply-templates/>
```

[11] Quelle: http://www.w3.org/TR/xslt#section-Document-Example (nicht „Erklärung zum Stylesheet“)

```
  </h1>
</xsl:template>

<xsl:template match="chapter/title">
  <h2>
    <xsl:apply-templates/>
  </h2>
</xsl:template>

<xsl:template match="section/title">
  <h3>
    <xsl:apply-templates/>
  </h3>
</xsl:template>

<xsl:template match="para">
  <p>
    <xsl:apply-templates/>
  </p>
</xsl:template>

<xsl:template match="note">
  <p class="note">
    <b>NOTE: </b>
    <xsl:apply-templates/>
  </p>
</xsl:template>

<xsl:template match="emph">
  <em>
    <xsl:apply-templates/>
  </em>
</xsl:template>

</xsl:stylesheet>
```

5.4.3 Erklärung zum Stylesheet:

1) Zuerst werden die verwendeten Namensräume mit ihren Kürzeln festgelegt. Zum einem ist die der XSL-Namensraum, der mit dem Kürzel „xsl“ versehen wird und zum anderen der XHTML1.0-Namensraum, der in diesem Dokument als der Default-Namespace fungiert und von daher kein Namensraumkürzel erhält. Damit geht der Prozessor davon aus, dass es sich bei allen Elementen, die er ohne Namensraumangabe findet, um Standard-HTML-Tags handelt.
2) Als nächstes wird festgelegt, dass Elemente mit den Namen „doc“, „chapter“ und „section“ in der Ausgabe ignoriert werden, sofern ihr Text nur aus Whitespace besteht.
3) Als Ausgabemethode wird festgelegt, dass XML erzeugt werden soll, wobei das Ausgabedokument eingerückt werden soll und der verwendete Zeichensatz der ISO-8859-1 (Latin1) sein soll. Der Grund für die Wahl von

„XML“ als Ausgabeformat ist der, dass damit die XML-Deklaration am Anfang der Ausgabe erzeugt wird, die für korrektes XHTML erforderlich ist.

4) Dann erfolgt die eigentliche Ausgabeerzeugung. Das erste Template ist für alle Elemente mit Namen „doc“ zuständig. Davon gibt es in diesem Fall genau eins. Es wird ein wenig HTML-Ausgabe erzeugt und der Titel des Dokuments ist der Wert, der am Knoten „title“ direkt unterhalb des aktuellen „doc“-Knotens steht.
5) Im <body>-Tag wird dann mit apply-templates der Prozessor auf die Suche nach passenden Knoten zu den vorhandenen Templates geschickt. Der nächste Treffer ist dann das <title>-Tag unterhalb des <doc>-Knotens und somit wird diese Regel abgearbeitet. Danach geht der Prozessor wieder auf die Suche usw.

5.4.4 Erzeugte Ausgabe:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<html xmlns="http://www.w3.org/TR/xhtml1/strict">
  <head>
    <title>Document Title</title>
  </head>
  <body>
    <h1>Document Title</h1>
    <h2>Chapter Title</h2>
    <h3>Section Title</h3>
    <p>This is a test.</p>
    <p class="note">
      <b>NOTE: </b>This is a note.
    </p>
    <h3>Another Section Title</h3>
    <p>This is <em>another</em> test.</p>
    <p class="note">
      <b>NOTE: </b>This is another note.</p>
  </body>
</html>
```

6 XML-Path Language (XPath)

6.1 Konzept

XPath bietet die Möglichkeit, sich gezielt durch ein XML-Dokument zu bewegen und bestimmte Knoten zu selektieren.
Dafür werden die sog. location steps verwendet, bei denen immer, von einem Kontext-Knoten (aktuell vom Prozessor verarbeiteter Knoten) ausgehend, relativ die gewünschten Schritte getätigt werden.
In der XPath-Logik besteht ein Knoten immer aus drei Elementen: der →Achse, dem →Knotentest und einem oder mehreren →Prädikaten.

6.1.1 Die Achse[12]

„Die Achse unterteilt ein Dokument bezüglich des Kontext-Knotens. Diese Achse dient der Definition einer Start-Umgebung für den Knoten-Test und die Prädikate bei der Auswertung von XPath-Ausdrücken."

Achse	**Beschreibung**
Child	Enthält alle Kind-Elemente (direkte Nachfahren) des Kontext-Knotens
Descendant	Enthält alle Nachfahren des Kontext-Knotens.
Parent	Das Eltern-Element des Kontext-Knotens.
Ancestor	Enthält alle Vorfahren des Kontext-Knotens.
Following-sibling	Alle nachfolgenden Geschwister-Elemente des Kontext-Knotens.
Preceding-sibling	Alle vorhergehenden Geschwister-Elemente des Kontext-Knotens.
Following	Alle Knoten, die dem Kontext-Knoten im Dokument nachfolgen. Diese Achse enthält weder die Nachfahren des Kontext-Knotens nach Attribute- und Namensraum-Knoten.
Preceding	Alle Knoten, die dem Kontext-Knoten im Dokument vorangegangen sind. Diese Achse enthält weder die Nachfahren des Kontext-Knotens nach Attribute- und Namensraum-Knoten.
Attribute	Die Attribut-Knoten des Kontext-Knotens
Namespace	Die Namensraum-Knoten des Kontext-Knotens.
Self	**Allein der Kontext-Knoten.**
Descendant-or-	**Die Vereinigungsmenge aus den Knoten der Nachfahren**

[12] Quelle: XML Professional, S. 330f

self	**und dem Knoten selbst.**
Ancestor-of-self	**Die Vereinigungsmenge aus den Knoten der Vorfahren und dem Knoten selbst.**

6.1.2 Abkürzende Schreibweise für Achsen

Da die Achsen in Dokumenten sehr viel verwendet werden (insb. in XSL), wurden Abkürzungen eingeführt, die die Handhabung und damit auch die Navigation deutlich vereinfachen. Im Folgenden sind diese aufgeführt:

Achse	**Abkürzung**	**Beispiel**
Child	**(keine Angabe)**	**/child::Book ➔ Book**
Attribute	**@**	**Book/attribute::color ➔ Book/@color**
Descendant-or-self	**//**	**Book/descendant-or-self::/Title ➔ //Title**
self	**.**	**self::node()//Title ➔ .//Title**
parent	**..**	**parent:node()/Title ➔ ../Title**

6.1.3 Knoten-Tests[13]

„Durch so genannte Knoten-Tests lassen sich spezifische Elemente der Knotentypen (Elemente, Texte, Kommentare, processing instructions) innerhalb der angegebenen Achse spezifizieren. Es gibt dabei verschiedenen Arten von Knoten-Test:

- Die Angabe eines Elementnamens sucht alle Knoten mit dem entsprechenden Namen heraus. Ein Knoten-Test für ‚Book' würde alle Elemente vom Typ <Book> innerhalb der angegebenen Achse zurückliefern.
- Das Metazeichen ‚*' fungiert als Joker und trifft auf alle Elemente innerhalb der Achse zu.
- Der Knoten-Test *node()* sucht alle Knoten in der Achse heraus
- Der Knoten-Test *text()* sucht alle Elemente mit dem reinen Text als Inhalt innerhalb der Achse heraus
- Der Knoten-Test *comment()* sucht alle Kommentar-Elemente innerhalb der Achse heraus.
- Der Knoten-Test *processing-instruction()* sucht alle Verarbeitungsanweisungen in der angegebenen Achse heraus. Zusätzlich kann ein Name innerhalb der Klammern angegeben werden, der dann die

[13] Quelle: XML Professional, S. 331

Suche auf die Verarbeitungsanweisungen mit dem entsprechenden Namen beschränkt."

6.1.4 Prädikate[14]

„Prädikate erlauben eine noch feinere Filterung der Suche nach Knoten. Ein Prädikat ist ein boolescher Ausdruck, der für jeden Knoten der Ergebnismenge aus dem Knoten-Test ausgewertet wird.

XPath bietet eine Reihe von Funktionen[15], die bei einem Test für diese Knoten angewendet werden können. Diese Funktionen geben verschiedenen Wertetypen zurück, darunter Strings und Zahlen, die man mit den Operationen =, !=, <=, <, >= und > auswerten kann. Umfangreichere Ausdrücken können auch durch die booleschen Operatoren *and* und *or* aufgespaltet werden. In diesem Fall werden beide Seiten des Ausdrucks ([…]) an die Funktion *Boolean()* übergeben, die dann den Ausdruck auf die folgende Art bearbeitet:

- Zahlen: nur wahr, wenn nicht null, positive null oder NaN (Not-A-Number)
- Knoten-Mengen: nur wahr, wenn die Menge nicht leer ist
- Strings: nur wahr, wenn die Länge größer als null ist
- Objekte: wenn ein Objekt nicht von einem der vier Grundtypen (Zahl, Knoten-Menge, boolescher Wert und String) ist, dann wird er in einen booleschen Wert konvertiert. Die Art der Konvertierung hängt von dem ursprünglichen Objekttyp ab."

[14] Quelle: XML Professional, S. 331

[15] s. http://www.w3.org/TR/xpath#corelib

7 XML-Query Language (XQuery)

7.1 Vorbemerkungen

Die XQuery-Definition hat derzeit (11/2005) noch nicht den Status einer Empfehlung des W3C. Die Arbeitsgruppe ist noch damit beschäftigt, die letzten Arbeiten an der Empfehlung vorzunehmen und Testergebnisse zu untersuchen. Aktuell handelt es sich bei XQuery um einen Recommendation Candidate.
Als weitere Vorbemerkung sei noch erwähnt, dass es sich bei XQuery nicht um eine XML-Sprache handelt. Auch wenn eine der Anforderungen an die Arbeitsgruppe war, dass die Syntax in XML verfasst sein sollte, so ist das nur ein Teil der Möglichkeiten von XQuery. Es soll nämlich in der Lage sein, verschiedenen Syntaxen anzunehmen. Eine davon ist XML. Die dadurch entstehende Sprache nennt sich XQueryX (XQuery in XML-Syntax).

7.2 Konzept

Bei XQuery handelt es sich um eine neue Abfragesprache, die (zumindest in der XML-Ausprägung) auf der XPath-Sprache basiert und die Datentypen aus XML-Schemas übernimmt. Die Sprache hört aber nicht da auf, wo XPath es getan hat, sondern zeigt sich wesentlich flexibler und intuitiver als ihr Vorfahre. Kurz- bis mittelfristig ist davon auszugehen, dass XQueryX aufgrund seiner Einfachheit und gleichzeitigen Mächtigkeit XPath ablösen wird, da dieses ja in XQuery integriert ist.
Der Anwendungsbereich der XQuery-Sprache liegt nicht nur in der Verarbeitung von XML-Dokumenten. Es sollen Dokumente für menschliche Leser, rein datenorientierte aber und auch Dokumente mit gemischtem (Text + eingebettete Daten) Inhalten verarbeitet werden können. Dabei wäre eine Vorstellbare Funktion die Indizierung oder Durchsuchung eines Textdokuments nach bestimmten Kriterien.
Bei genauerer Betrachtung der Syntax wird man aber auch feststellen, dass eine gewisse Ähnlichkeit mit SQL vorhanden ist.

7.3 Anforderungen an die W3C-XML-Query-Group

Im Folgenden werden einige Anforderungen aufgezeigt, die die Arbeitsgruppe des W3C für die Entwicklung des neuen Standards berücksichtigen mussten:

- Syntaxanforderungen
    - Die Syntax muss leicht für Menschen lesbar sein

    - Mehrere Syntaxausprägungen können möglich sein
    - Eine Abfragesprache muss in XML-Syntax verfasst sein
- Protokollunabhängigkeit von XML muss gewahrt bleiben
- Erzeugung von definierten Fehlerzuständen bei der Verarbeitung muss möglich sein
- Aktualisierbarkeit der Definition muss möglich sein

Das sind nur einige der Anforderungen, auf die die Arbeitsgruppe Rücksicht nehmen musste. Von den eben genannten Bedingungen wurden alle in den bisherigen Stand der „Empfehlung“ eingebaut.

7.4 Ausdrucksformen in XQuery

In XQuery stehen (wie auch in anderen Programmiersprachen) verschiedenen Formen von Ausdrücken zur Verfügung.

- Pfad Ausdrücke.Hierbei handelt es sich um normale XPath-Ausdrücke.
- FLWOR-Ausdrücke. Diese Gruppe der Ausdrücke bildet eine Art SQL für XML-Dateien. Die Abkürzung FLWOR („flower“) steht für For-Let-Where-Order-Return.
- Listenausdrücke. Hierbei werden Listen von Werten bearbeitet. Zu diesem Zweck stehen verschiedenen Funktionen zu Verfügung[16]
- Bedingte Ausdrücke. Bekannte if-then-else Ausdrücke, die die Verarbeitung an Bedingungen knüpfen.
- Quantifizierte Ausdrücke prüfen, ob eine Menge von Elementen/Werten eine Bedingung erfüllt.
- Datentyp-Ausdrücke

7.4.1 FLWOR-Beispiel

```
xquery version "1.0";

<beispiel>
  <bsp1>
  {
  for $d in doc("vortrag.xml")/präsentation/inhalt//kapitel
  where count($d/absatz) >= 5
  order by $d/@nummer descending
  return
   <kapitel überschrift="{$d/@titel}">
      {
      for $absatz in $d//absatz
      where $absatz/überschrift != ''
```

[16] http://www.w3.org/TR/2005/CR-xpath-functions-20051103/

```
      return
        <thema>{$absatz/überschrift/text()}</thema>
      }
    </kapitel>
  }
  </bsp1>
</beispiel>
```

Das Beispiel sucht für alle Kapitel, die mehr als fünf Absätze haben die Überschriften heraus und liefert dann für jedes Kapitel die Absatzüberschriften, wenn diese vorhanden sind. Das Ergebnis wird dann in den Knoten <bsp1> geschrieben. Die Ausgabe für das Dokument (vortrag.xml) sieht dann folgendermaßen aus.

```
<beispiel>
  <bsp1>
    <kapitel überschrift="XML-Path Language (XPath)">
      <thema>Konzept</thema>
      <thema>Achse</thema>
      <thema>Abkürzende Schreibweisen:</thema>
      <thema>Knoten-Tests</thema>
      <thema>Prädikate</thema>
      <thema>Beispiele</thema>
    </kapitel>
    <kapitel überschrift="XML-Query Language (XQuery)">
      <thema>Konzept</thema>
      <thema>Anforderungen an W3C XML-Query Group</thema>
      <thema>Ausdrucksformen in XQuery</thema>
      <thema>FLWOR-Beispiel</thema>
    </kapitel>
    <kapitel überschrift="XML-Schema Definition Language (XSD)">
      <thema>Kennzeichen von Schema</thema>
      <thema>Schema-Deklaration</thema>
      <thema>Elementdeklaration</thema>
      <thema>vordefinierte Elemente (Auswahl)</thema>
      <thema>simpleType</thema>
      <thema>Datentypen (Auswahl)</thema>
      <thema>complexType</thema>
    </kapitel>
    <kapitel überschrift="DocumentType Definition (DTD)">
      <thema>Konzept</thema>
      <thema>Syntax</thema>
      <thema>Operatoren</thema>
      <thema>einfache Elemente</thema>
      <thema>komplexe Elemente</thema>
      <thema>Attribute</thema>
      <thema>Vorgabewerte</thema>
      <thema>Attributypen</thema>
      <thema>ENTITY</thema>
    </kapitel>
    <kapitel überschrift="Extensible Markup Language (XML)">
      <thema>Wofür steht XML?</thema>
      <thema>Wie sieht XML aus?</thema>
      <thema>Dokumentgliederung</thema>
      <thema>Elemente</thema>
      <thema>Attribute</thema>
      <thema>Entity-Referenzen</thema>
      <thema>Vordefinierte Entities</thema>
      <thema>Verarbeitungsanweisungen (processing instructions)</thema>
      <thema>Kommentare</thema>
    </kapitel>
```

```
  </bsp1>
</beispiel>
```

7.5 Fortschritte gegenüber XPath und XSL

Durch XQuery ist es einfacher und vor allem auf kleinerem Raum möglich, gleicher Ergebnisse wie mit XSL + XPath zu erreichen. Darum ist davon auszugehen, dass XQuery diese beiden Standards in naher Zukunft ablösen wird. Der Vorteil von XQuery ist außerdem, dass die Datenübertragung verringert werden kann. Für XSLT ist es nötig, dass das gesamte XML-Dokument übertragen wird, bevor die Umwandlung beginnt. Mit XQuery hat man die Möglichkeit, auf die entfernte Quelle zuzugreifen und nur das Dokumentfragment zu übertragen, das von Interesse ist. Als Beispiel sei ein Produktkatalog im XML-Format genannt. Wenn man nur einen bestimmten Teil des Katalogs darstellen möchte, kann mit einer einfachen Abfrage (wie aus einer Datenbank) dieser Bedarf gedeckt werden, ohne dass der Gesamtkatalog übertragen werden muss.
Außerdem ist es mit den eingebauten Funktionen einfacher möglich, bestimmte Dokumente zu durchsuchen und somit eine Volltextsuche nach den Wünschen eines Anwenders zu implementieren.

8 Literaturverzeichnis

8.1 Bücher:

Anderson, Richard: XML Professional, 1. Auflage 2000, MITP Verlag GmbH, Bonn

Erl, Thomas: Service-Oriented Architecture, April 2004, Prentice Hall PTR

8.2 Webseiten:

http://www.w3.org/TR/2005/CR-xquery-20051103/, versionierte Seite

http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/, versionierte Seite

http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/ , versionierte Seite

http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/, versionierte Seite

http://www.brics.dk/~amoeller/XML/querying/index.html, 2005-11-16