Berufsakademie Mosbach
- Staatliche Studienakademie -
Lohrtalweg 10

74821 Mosbach

Transformationen von XML-Dokumenten mit XSL

Firma / Anschrift	dsb AG Konrad-Zuse-Straße 16 74172 Neckarsulm
Fachrichtung	Wirtschaftsinformatik
Name	Timo Steidle
Anschrift	Pappelweg 5 74177 Bad Friedrichshall
Abgabedatum	28.04.2006

Die Praxisarbeit ist formal und inhaltlich geprüft und zur Einreichung an die Berufsakademie freigegeben!

Unterschrift Ausbildungsbeauftragter

Stempel des Ausbildungsbetriebs

Inhaltsverzeichnis

1 Vorwort

Seit der Einführung Ende der 90er Jahre ist XML (eXtensible Markup Language) in aller Munde. XML musste von jedem Produkt in irgendeiner Weise unterstützt werden, egal wie nutzlos diese Funktionen auch waren. Durch diesen teilweise inflationären Einsatz von XML wurden die wirklichen Vorteile und die Funktionalität in den Hintergrund gestellt, oftmals wurde es falsch eingesetzt. Dadurch sind die wahren Einsatzmöglichkeiten und das Grundwesen von XML vielen Entwicklern und Analytikern noch nicht bekannt. Im Gegenteil, durch den entstandenen Hype und die vielfach sinnfreie Implementierung von XML-Funktionen besitzen sie eine unbegründete Ablehnung gegenüber XML.

Der künstliche und übertriebene Hype rund um XML verflachte in den letzten Jahren merklich, was der Entwicklung von XML sicher keinen Dämpfer bereitet hat, im Gegenteil. Entwickler entdecken nun den wahren Charakter von XML, dessen Konzept und dessen Stärken.

Doch offensichtlich haben Informatiker eine Vorliebe für das „X“ entwickelt. Denn kaum hat sich die Hysterie um XML gelegt, scheint sich nun ein weiterer Hype rund um ein „X“ zu erheben. Eng verknüpft mit XML hört man immer öfters von XSL und XSLT, manchmal auch als Synonyme füreinander. Sicher ist diese Euphorie nicht unbegründet, diese beide Akronyme werden noch für viel Furore sorgen, aber bevor diese Euphorie ins Negative umschlägt, versucht diese Arbeit, die Begriffe zu konkretisieren.

Das Ziel dieser Praxisarbeit ist sicher nicht, alle Einsatzmöglichkeiten und Funktionen von XML zu beschreiben und festzuhalten. Doch soll sie einen Überblick über das Konzept von XML bieten und dem Leser einen ersten Eindruck von den Einsatzmöglichkeiten und den vielfältigen Anwendungsgebieten vermitteln. Eine Stärke von XML wird hierbei genauer betrachtet und ausführlicher beschrieben, und zwar XSLT (eXtensible Stylesheet Language). Natürlich kann im Rahmen dieser Praxisarbeit kein Detailwissen vermittelt werden, noch möchte diese Arbeit als vollständige Referenz dienen, dafür gibt es genügend andere Literatur und Quellen. Sie gibt eine kurze Einführung in die Welt von XML und XSLT und ermöglicht dem Informatiker eine reelle Einschätzung der Stärken und Einsatzmöglichkeiten von XML beziehungsweise XSLT, abseits den Versprechungen der Marketing-Strategen.

2 Was ist XML?

2.1 Das Konzept von XML

Die eXtensible Markup Language (XML) ist ein Konzept zur Speicherung und Darstellung von Daten. Dabei sollen die Dokumente maschinen- und menschenlesbarer sein, im Vergleich zu anderen Konzepten. XML gibt dabei die Regeln für solche Dokumente vor. Die Grundidee dahinter ist, dass Dateien, die von Programmen gespeichert wurden, nicht nur von denselben Programmen wieder geöffnet und verarbeitet werden können. So können Microsoft Word Dokumente nur von Microsoft Word geöffnet werden, oder nur mit hohem Aufwand von Drittprogrammen interpretiert werden. Ein gutes Gegenbeispiel sind HTML-Dokumente. Diese sind für jeden frei zugänglich. Solch ein offener Standard ermöglicht der Softwareindustrie das Entwickeln von beliebig vielen Programmen, die Dokumente des selben Formates lesen, schreiben und interpretieren können. Dadurch ist der Benutzer nicht mehr an ein einziges Produkt gebunden, sondern kann zwischen vielen Alternativen auf dem Markt wählen und so das für sich passende Produkt aussuchen und benutzen.

Das wohl stärkste Argument für einen offenen Standard ist also die Unabhängigkeit des Benutzers an eine bestimme Software. Doch die Entwicklung der elektronischen Datenverarbeitung hat einen neuen wichtigen Aspekt hervorgebracht: die flexible Darstellung der Daten in verschiedenen Medien und Ausgabeformaten. So wird eine Rechnung des Kunden auf der Website anders gestaltet wie die Print-Ausgabe für die Archivierung. Die Produktdatenbank muss für den Browser in HTML konvertiert werden und eine Pflege der Produkte durch einen Angestellten sollte durch ein Programm ermöglicht werden.

Wie wir in den oberen Beispielen sehen, ist es wichtig, dass man Informationen in einer Vielzahl von Ausgabeformaten darstellen kann. Diese Flexibilität erreicht man durch die strikte Trennung von Daten und Gestaltungselementen. Also wenn die Daten strukturiert vorliegen und die Darstellung von der Software oder einer Schnittstelle vorgenommen werden.

Genau hier setzt XML an. XML ist eine Auszeichnungssprache und trennt somit Daten und Semantik voneinander. Die Daten werden dabei mithilfe von sogenannten Tags in einen semantischen Zusammenhang gesetzt.

Für XML gibt es auch Kontrollmechanismen wie die Dokumenttyp-Definition (DTD), durch die man grammatische Regeln für XML-Dokumente beschreiben kann, die ein Dokument einzuhalten hat. Mit so einer Dokumenttyp-Definition können dann geschrieben XML-Dokumente leicht validiert werden und auf ihre Gültigkeit überprüft werden. Dazu aber später mehr.

Die Grundidee von XML, Daten und Semantik voneinander zu trennen, wurde schon früher verfolgt und ist keine neue Erfindung von XML. Schon in den 60er Jahren des 20. Jahrhunderts gab es das Konzept des „generic coding“, das Beschreiben einer logischen Struktur ohne Berücksichtigung des Aussehens eines Dokumentes. Dies resultierte dann 1989 in der SGML, Standard Generalized Markup Language.
Da sich SGML aufgrund der Komplexität nie durchsetzen konnte, hat man es schlanker gemacht und unter dem Namen XML veröffentlicht. Somit ist XML ist eine Untermenge von SGML.

Durch XML könnte es in Zukunft nicht mehr bestimmte Programme für bestimme Dateiformate geben, sondern Programme für verschiedene Funktionen.

2.2 Strukturen eines XML-Dokumentes

XML-Dokumente sind reine Textdateien. Also reicht ein einfacher Editor schon, um XML-Dokumente zu erstellen, wie zum Beispiel der von Microsoft mitgelieferte Editor von Windows. Um eine erste Vorstellung vom Aufbau eines XML-Dokumentes zu bekommen, sollte man sich die Struktur als eine Baumstruktur vorstellen. Der Baum besteht aus den oben schon genannten Tags, welche die Daten auszeichnen, und den Daten selbst, welche als gewöhnliche Zeichenketten auftauchen.

Das Beispiel 2.2.1 zeigt ein einfaches, kleines XML-Beispiel.

```
1  <?xml version="1.0">
2  <Buchliste>
3          <Buch id="1">
4                  <Titel>Wien wartet auf dich!</Titel>
5                  <Autor>
6                          <Vorname>Tom</Vorname>
7                          <Nachname>DeMarco</Nachname>
8                  </Autor>
9                  <Autor>
```

```
10                          <Vorname>Timothy</Vorname>
11                          <Nachname>Lister</Nachname>
12                  </Autor>
13                  <ISBN>3-446-16229-1</ISBN>
14          </Buch>
15 </Buchliste>
```

Beispiel 2.2.1 – das erste XML-Dokument

Eine Baumansicht des Beispiels ist im Anhang hinterlegt[1]. Inhaltlich ist dieses Beispiel sicher nicht sehr anspruchsvoll, doch das verwendete XML ist ohne Fehler. Denn im Gegensatz zu anderen Auszeichnungssprachen wie HTML kann man seine Tags benennen, wie man möchte. Es steht jedem Entwickler also frei, sein Dokument so zu strukturieren, wie er es möchte. XML ist es egal, wie das Dokument später genutzt wird oder was die einzelnen Elemente bedeuten. Dies bedeutet aber dennoch, dass man als Entwickler bestimmte Grundregeln einhalten muss. Auf diese Grundregeln wird im nächsten Kapitel näher eingegangen.

2.2.1 Aufbau

Am Beispiel 2.2.1 wird deutlich, wie ein Tag auszusehen hat und was er bedeutet. Ein Tag besteht aus 2 Winkelklammern (< und >) und die von ihnen umschlossenen Namen. Grundsätzlich muss jedes Tag durch ein entsprechendes Endtag beendet werden. Dies hat ein „/“ vor dem Namen stehen. Also im obigen Beispiel wäre <Autor> ein Anfangstag und </Autor> das zugehörige Endtag. Zwischen den beiden Tags stehen dann die reinen Informationen des Dokuments. Hier in diesem Beispiel gibt es die Informationen Vornamen, Nachnamen etc.

Die erste Zeile bestimmt die Version von XML, welche für dieses Dokument verwendet wird. In diesem Beispiel wird also die Version 1.0 benutzt. Die Empfehlungen werden vom W3C[2] herausgebracht und weiterentwickelt. So kann man auf ihrer Seite auch die neuste Edition der Version 1.0 finden und ihre Empfehlungen nachlesen[3].

Außer diesem Informationselement besteht ein Dokument noch aus einer geordneten Liste von sogenannten Knoten. Der erste Knoten verkörpert immer den Startknoten und wird auch als Wurzel des Dokumentes bezeichnet. Dieses Startknoten beinhaltet nun

1 Siehe Anhang 5.1
2 World Wide Web Consortium – Gremium zur Standardisierung von Web-Technologien.
3 XML 1.0 (Third Edition) - http://www.w3.org/TR/2004/REC-xml-20040204/

Kindelemente in Form von weiteren Knoten.

In unserem Beispiel ist also das Element „Buchliste“ die Wurzel. Nun besitzt diese Bücherliste natürlich auch Bücher, welche jeweils als „Buch“-Knoten repräsentiert werden. Nun stehen als weitere Knoten auch die Informationen zum Buch im Dokument, teilweise auch wieder mit Kindelementen, wie der Knoten „Autor“ mit „Vorname“ und „Nachname“. Um ein weiteres Buch in die Liste aufzunehmen, benötigt man nur ein weiteren „Buch“-Knoten unterhalb der „Buchliste“.

2.2.2 DTD – Dokumenttyp-Definition

Im Zusammenhang von XML und XML-Dokumenten fallen oft die Begriffe „wohlgeformt“ und „gültig“. Um sein Dokument wohlgeformt nennen zu können, reicht es aus, die syntaktischen Regeln einzuhalten. Dazu gehört eine XML-Deklaration in der ersten Zeile des Dokumentes und ein Wurzelelement, welches den Inhalt umschließt.

Oftmals reicht es aber nicht aus, nur die Syntax eines Dokumentes zu überprüfen. Um auch Tagnamen und den strukturellen Aufbau zu prüfen, kann man ein Regelwerk formulieren. Die Dokumenttyp-Definition ist genau so ein Regelwerk, mit dem man Dokumente auch strukturell überprüfen kann. Ein Dokument ist demnach „gültig“, wenn es auch die Prüfung durch die DTD besteht.

Eine Dokumenttyp-Definition kann entweder direkt in das Dokument selbst geschrieben werden oder, im Normalfall, in eine weitere Datei ausgelagert werden. Sinnvoller ist das Auslagern der DTD in eine Datei, da die DTD in diesem Fall nur einmal angelegt werden muss, um sie für einen beliebig großen Bestand zu benutzen. Innerhalb der Dokumente muss man nur noch auf diese Dokumenttyp-Definition verweisen. Dieser Verweis sieht folgendermaßen aus:

```
1  <?xml version="1.0" ?>
2  <!DOCTYPE Buchliste SYSTEM "Buchliste.dtd">
3  <!-- restliche Daten -->
```

Beispiel 2.2.2.1 – Verweis einer DTD

Solche Dokumenttyp-Definitionen sind selbst auch reine Textdateien und können so mit jedem beliebigen Editor bearbeitet und erstellt werden. Natürlich gibt es inzwischen auch

Programme, welche die Dokumenttyp-Definitionen für den Entwickler ausformulieren und erstellen.

2.2.3 XML Schema

Die Dokumenttyp-Definition wurde lange Zeit benutzt, um seine Dokumente auf Gültigkeit zu überprüfen. Ein Grund war, dass dies der einzige Weg war. Doch die DTD unterstütze einige Funktionen nicht. So kennt die DTD keine Datentypen, noch entspricht eine DTD selbst nicht der XML-Syntax.

Aus diesen und vielen anderen Gründen wurde XML Schema entwickelt. XML Schema ist eine weitere Möglichkeit, seine XML-Dokumente zu validieren. Die Syntax ist XML-konform und somit leicht zu erlernen. Daneben bietet XML Schema Datentypen, feinere Einstellungsmöglichkeiten und viele weitere Funktionen, die das Arbeiten erleichtern. Auf XML Schema wird hier aber nicht näher eingegangen, da dies den Rahmen dieser Praxisarbeit sprengen würde.

3 XSLT in der Praxis

Nun hat man zwar ein wohlgeformtes und optional auch ein gültiges XML-Dokument, doch scheint dies noch sehr statisch zu sein. Man kann schlecht das XML-Dokument als Website im Browser präsentieren oder es so für den Druck ausgeben. Doch gerade die reine Abbildung der Daten als Text-Elemente ist ein Garant für die Wiederverwendbarkeit der Daten. Während zum Beispiel Microsoft Word Dokumente sehr stark auf Layout- und Präsentationsinformationen baut, erkennt man bei diesen Dateien kaum eine Struktur. Eine Anpassung des Inhalts hat meistens die Zerstörung der Formatierung zur Folge.

Gerade hier offenbart nun XML eine seiner Stärken. XML-Dokumente lassen sich durch sogenannte Transformations von einer Form in eine andere Form umwandeln. Dabei können Operationen, wie zum Beispiel Fallunterscheidungen oder Schleifen, angewendet werden. So können sie auf der einen Seite ihr XML-Dokument in den Bearbeitungs-prozess geben, auf der anderen Seite kommt dann mit Hilfe der Transformations ein komplett anderes Dokument raus, zum Beispiel PDF, XHTML, reiner Text oder selbst wieder ein XML-Dokument.

3.1 Was ist XSLT?

Um solche Transformations realisieren zu können, empfiehlt sich XSLT („eXtensible Stylesheet Language Transformation“). XSLT ist eine Teilmenge von XSL („eXtensible Stylesheet Language“). XSL besteht aus 3 wesentlichen Komponenten. Einerseits gibt es XSL Formatting Objects (XSL-FO), um mit Stilangaben und Formatierungsanweisungen ein Dokument zu beschreiben. Andererseits gibt es noch XPath, um die Adressierung von Baumbestandteile effizient und umfangreich realisieren zu können. Da diese beiden Komponenten aber selbst komplex und umfangreich sind, werden sie in dieser Arbeit nicht weiter betrachtet.

XSLT ist also die dritte Komponente, aus denen XSL besteht. Mit Hilfe von XSLT können sie Anweisungen definieren, um die Informationen eines XML-Dokumententyps in einen anderen Dokumenttyp umzuwandeln. Im Folgenden werden Probleme genannt und deren mögliche Lösung mithilfe XSLT beschrieben.

Oftmals bekommt man Dokumente, mit dessen Format nicht weitergearbeitet werden

kann. Nun kann ein XSLT-Stylesheet die erforderlichen Informationen in das gewünschte Format transformieren.

Viele Dokumente sind voller Informationen, die man nicht braucht und die das Dokument unnötig aufblasen. Auch hier kann ein XSLT-Stylesheet behilflich sein und ein schlankes Dokument generieren, welches nur die relevanten Elemente enthält.

Heutzutage möchte jeder das Internet nutzen und Dokumente im Web veröffentlichen. Doch der Dokumenttyp der Dokumente ist oftmals zu speziell und komplex für die Darstellung. Cascading Stylesheets (CSS) sind für diese Dokumente nicht geeignet. Mithilfe von XSLT-Stylesheets können die Dokumente leicht in sauberes XHTML transformiert werden und so im Web dargestellt werden.

Die oben beschriebenen Probleme können sicher auch mit mehr oder weniger aufwendiger Programmierung gelöst werden, doch ist diese Programmierung oftmals mit einem vergleichbar höherem Aufwand verbunden, wie die Lösung mit XSLT. Zudem wurde XSLT genau für diese Aufgabe konzipiert und ist dazu noch leicht zu erlernen, da die Syntax den Regeln von XML folgt.

3.2 Vorgehensweise von XSLT

Die Programmiersprache XSLT beschreibt also, wie ein XML-Dokumenttyp in ein anderes Dokument transformiert werden soll. Dabei gehorcht das generierte Dokument meist auch der XML-Syntax, was aber nicht immer der Fall sein muss. Mit XSLT lassen sich auch zum Beispiel Binärdateien oder Textdateien erstellen.

Um die Transformationen durchführen zu können, werden beide Dokumente, Quelldatei und Zieldatei, als logische Bäume betrachtet. Regeln und Anweisungen sind nötig, um eine Transformation geregelt ausführen zu können. Diese Regeln werden werden bei XSLT Templates genannt, im Deutschen hat sich der Begriff Vorlage dafür noch nicht durchgesetzt. Jede Vorlage hat ein Muster, welches mit XPath ausgedrückt werden kann. Mit diesem Muster wird dann festgelegt, für welche Knoten des Quelldokumentes diese Vorlage gilt. Innerhalb einer solchen Vorlagen steht dann beschrieben, wie sie den Knoten für die Zieldatei zu transformieren hat.

Das folgende Code-Beispiel zeigt eine solche Vorlage:

```
1  <xsl:template match="/Buchliste/Buch">
2    <h1>
3      <xsl:value-of select=”Titel”/>
4    </h1>
5  </xsl:template>
```

Beispiel 3.2.1 – eine XSLT-Vorlage

Das oben erwähnte Muster ist in diesem Beispiel die Zeichenkette „Buch“. Auf jeden gefundenen Buch-Knoten wird also dieses Template - diese Vorlage – angewendet. Da dies eine Regel für die Transformation ist, wird dies auch Vorlagen-Regel oder „Template Rule“ genannt. Sollte also bei der Transformation ein Buch-Knoten auftauchen, wird diese Vorlage verwendet, welche schlicht den Titel des Buches als Überschrift zwischen zwei HTML-Tags ausgibt.

Dieses kleine Beispiel soll nur einen ersten, kurzen Eindruck von der Syntax und der Arbeitsweise von XSLT wiedergeben. Im nächsten Kapitel wird dann näher auf die Struktur und die Syntax von XSLT eingegangen.

3.3 Strukturen eines XSLT-Dokumentes

Wie oben schon erwähnt, sind XSLT-Dokumente nichts anderes als wohlgeformte XML-Dokumente, welche aus den Elementen der Sprache XSLT bestehen. Also benötigt man zum Schreiben von XSLT nur einen einfachen Texteditor. Natürlich gibt es inzwischen auch viele andere Hilfsmittel, die das Erstellen vereinfachen, aber nicht zwingend notwendig sind.

3.3.1 Aufbau

Um nun den Anforderungen eines wohlgeformten XML-Dokumentes gerecht zu werden, benötigt man in seinem XSLT mindestens eine Versionsangabe und einen Wurzelknoten. Auf die Versionsangabe wurde schon in Kapitel 2.2.1 hingewiesen. Die Angabe wird also in einem sogenannten Informationselement angegeben. Dabei wird bestimmt, welche XML-Version verwendet werden soll.

Des weiteren benötigt man ein Wurzelknoten. Dieser Wurzelknoten kennzeichnet immer den Start des Baumes und umgibt den gesamten Inhalt des Dokumentes. Innerhalb dieses Wurzelknotens werden nun die Regeln für die Transformation mithilfe von

Elementen beschrieben. Das erste Element ist also eine Art Container. Im Listing 3.3.1.1 erkennt man, wie dieses Element auszusehen hat. Jedes XSLT-Element hat ein *xsl* als Präfix, welches durch einen Doppelpunkt vom Namen getrennt wird. Das Wurzelelement heißt also *stylesheet*. Alternativ kann es auch *transform* genannt werden. Jedoch wird diese Bezeichnung nicht oft verwendet, obwohl sie die gleiche Bedeutung besitzen.

```
1  <?xml version="1.0" ?>
2  <xsl:stylesheet version="1.0“ xmlns:xsl="http://www.w3.org/1999/XSL/Transform“>
3          <!-- Transformations-Logik -->
4  </xsl:stylesheet>
```

Listing 3.3.1.1 – erste Zeile eines XSLT-Dokumentes

In diesem kurzen Coding werden auch die zwei wichtigsten Attribute von *xsl:stylesheet* verwendet. Zum einen ist dies auch wieder die Versionsangabe, welche hier die benutzte Version von XSLT angibt. Das zweite Attribut bestimmt den Namensraum. Dazu im nächsten Kapitel mehr.

3.3.2 Namensraum

Der Namensraum wird als eindeutige Adresse angegeben. Zusätzlich kann diese Angabe noch durch ein Namensraumpräfix erweitert werden (bei Listing 3.3.1.1 ist es *xsl*). Dieses Präfix symbolisiert dann den Namensraum. Der Namensraum definiert die Elemente und Attribute und grenzt sie von anderen Strukturen ab. In einem XSLT-Dokument wird der Namensraum mithilfe des *xmlns*-Attributs innerhalb des Wurzelelements angegeben. Das Präfix wird dann im ganzen Dokument vor die Elemente gesetzt.

3.3.3 Entwicklung

XSLT wird vom W3C weiterentwickelt. Auf deren Homepage kann man auch den Stand von neuen Versionen abrufen und sich einen Überblick über den Entwicklungsprozess der Sprache verschaffen. Die aktuelle Version ist 1.0[4], aber die neue Version 2.0 wird wohl bald offiziell veröffentlicht werden. Die ersten Eindrücke über 2.0 kann man aber schon beim „Candidate Recommendation“ sammeln[5].

4 XSLT 1.0 - http://www.w3.org/TR/xslt
5 XSLT 2.0 - http://www.w3.org/TR/xslt20/

3.4 Ein erstes Beispiel

Um die Funktionen und Elemente von XSLT besser verstehen zu können, betrachten wir sie uns anhand eines Beispiels näher. Dazu wurde das Beispiel aus Kapitel 2.2 (Beispiel 2.2.1 - „das erste XML-Dokument“) um weitere Datensätze erweitert. Das neue Beispiel ist im Anhang hinterlegt. Wie man erkennen kann, besteht das XML-Dokument wieder aus dem Wurzelknoten „Buchliste“. Diesmal besitzt dieser Knoten jedoch zwei Kindelemente, zwei „Buch“-Knoten. Das erstellte Dokument soll am Ende ein HTML-Dokument sein, da HTML leicht zu lesen ist und man das Ergebnis später auch leicht im Browser anschauen kann. Den Code der XML-Datei ist im Anhang hinterlegt.

3.4.1 Vorlagen

Vorlagen sind die Grundzutat jeder XSL-Transformation, denn die Vorlagen (Templates) beinhalten die Transformationsregeln. Ein erstes Beispiel wurde schon in Kapitel 3.2 beschrieben. Das Element ist also folgendermaßen aufgebaut:

```
1  <xsl:template match="/">
2   </xsl:template>
```

Beispiel 3.4.1.1 – eine leere Vorlage

Hier ist nur das wichtigste Attribut angegeben, *match*. Mithilfe dieses Attributs kann man genau festlegen, über welchen Ausdruck diese Vorlage aufrufbar ist. Gibt man dort „/“ an, so gilt diese Vorlage für das Wurzelelement. Unter 5.2.2 sieht man solch eine Vorlage. Übergibt man nun das XML-Dokument und die XSLT-Datei einem Parser, bekommt man folgendes HTML-Coding zurück:

```
1  <html>
2          <head>
3                  <title>Bücherliste</title>
4          </head>
5          <body>
6                  <h1>Die Bücherliste</h1>
7          </body>
8  </html>
```

Listing 3.4.1.1 – Ergebnis der Transformation

Wie man erkennen kann, wird nur das Wurzelelement transformiert und durch diesen Code ersetzt. Weitere Transformationsregeln wurden noch nicht eingebaut.

3.4.2 Wiederholungen

Nun wollen wir natürlich noch unseren Bücherbestand mit ausgeben. Da wäre es natürlich von Vorteil, wenn XSLT auch Schleifen unterstützt. Für Wiederholungen verwendet man

in XSLT das Element *for-each*. Das Element enthält ein Attribut namens *select*, welches den Ausdruck enthält, für den eine passende Vorlage gesucht wird. Das Beispiel wurde um dieses Element erweitert und steht nun im Anhang unter 5.2.3.

Das HTML wurde um eine Liste erweitert (*ul*), jedes Buch wird dann als Listenpunkt ausgegeben. Die Anweisungen der *for-each* – Schleife werden sooft wiederholt, bis jedes Buch innerhalb der Buchliste abgearbeitet wurde. Die Anweisungen innerhalb der Schleife beinhalten auch 2 neue Elemente. Mit *value-of* wird eine Information aus dem Dokument ausgegeben. Auch dieses Element hat *select* als mögliches Attribut. Hier wird ebenso der Ausdruck angegeben, auf welchen diese Anweisung anspringt. Einmal wird der Titel ausgegeben, einmal die ISBN-Nummer und einmal der Nachname des Autoren.

Da das Format, welches generiert wird, auch der XML-Syntax untergeordnet ist, kann nicht ohne weiteres reiner Text ausgegeben. Darum muss auf das Element *text* ausgewichen werden. Die Handhabung ist aber denkbar einfach. Der gewünschte Text wird einfach zwischen die beiden Tags gesetzt. Die Ausgabe sieht folgendermaßen aus:

```
1  <html>
2          <head>
3                  <title>Bücherliste</title>
4          </head>
5          <body>
6                  <h1>Die Bücherliste</h1>
7                  <ul>
8                          <li>Wien wartet auf dich! - DeMarco - 3-446-16229-1</li>
9                          <li>Einführung in XML - Ray - 3-89721-286-2</li>
10                         <li>Mythos Motivation - Sprenger - 3-593-37637-7</li>
11                 </ul>
12         </body>
13 </html>
```

Listing 3.4.2.1 – Ergebnis mit Schleife

3.4.3 Fallunterscheidungen

In anderen Programmiersprachen werden Fallunterscheidungen oft durch *if* realisiert. Dies ist in XSLT nicht anders. Ein Element für die Fallunterscheidungen heißt ebenfalls *if* und lässt sich analog zu anderen Sprachen verwenden. Im Anhang 5.2.4 wurde das bisherige XSLT erweitert um eine if-Abfrage. Dazu musste nur das Element *if* eingefügt werden, welches ein Attribut *test* hat. Innerhalb dieses Attributs wird dann die Fallunterscheidung vorgenommen. In diesem Beispiel wird getestet, ob das Attribut *Type* entweder 't' oder 'g'

ist. Diese Angaben bestimmen, ob es sich um ein Taschenbuch handelt oder um eine gebundene Ausgabe. Dementsprechend wird ein anderer Text ausgegeben. Mithilfe des @-Zeichens kann man auf Attribute zugreifen. Die Ausgabe ergibt folgendes:

```
1  <html>
2          <head>
3                  <title>Bücherliste</title>
4          </head>
5          <body>
6                  <h1>Die Bücherliste</h1>
7                  <ul>
8                          <li>Wien wartet auf dich! - Taschenbuch</li>
9                          <li>Einführung in XML - gebundene Ausgabe</li>
10                         <li>Mythos Motivation - gebundene Ausgabe</li>
11                 </ul>
12         </body>
13 </html>
```

Listing 3.4.3.1 – Ausgabe nach Fallunterscheidung

Mit dem Element *if* kann man nur einfache Fallunterscheidungen ohne Sonderfälle durchführen. Um zum Beispiel auch *elseif* oder *else* verwenden zu können, benötigt man weitere Elemente - aufgrund der Fülle an Funktionen wird auf sie nicht näher eingegangen.

4 Fazit

Durch die Benutzung von XML erhält man viele Vorteile und Erleichterungen, die man, einmal entdeckt, nicht mehr missen möchte. Auch den wichtigsten Anforderungen neuer Datenformate kann XML gerecht werden. XML-Dokumente können ohne viel Aufwand auf verschiedensten Medien dargestellt werden und auch die Darstellung selbst kann leicht manipuliert und geändert werden. Diese Anforderungen werden dabei durch XSL realisiert und umgesetzt. Die oben beschriebene Sprache XSLT kümmert sich dabei um die Transformation von XML-Dokumente in andere Dokumente.

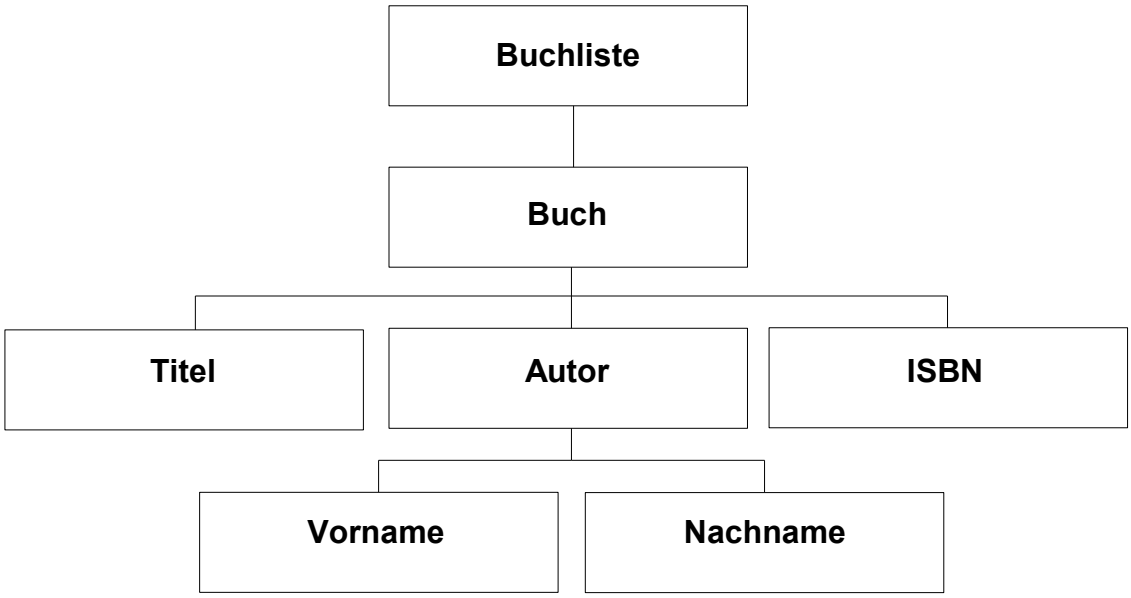
Bisher brauchte man eine vollwertige Programmiersprache, um Dokumente zu transformieren. Um individuelle Transformationen durchzuführen, musste man neue Programme schreiben, was viel Aufwand und somit auch Zeit gekostet hat. Dabei waren die verwendeten Programmiersprachen überladen an Funktionen, die man nicht braucht und somit den ganzen Prozess verlangsamen.

Mit XSLT wurde eine Sprache entwickelt, deren Syntax und Arbeitsweise leicht zu erlernen ist und die selbst auch den syntaktischen Regeln entspricht, welche man schon kennt – den Regeln von XML. Trotz der Einfachheit und dem schlanken Auftreten von XSLT, kennt die Sprache Funktionen wie Fallunterscheidungen und Schleifen. Somit stehen dem Entwickler alle Möglichkeiten zur Verfügung, seine Dokumente zu transformieren.

Aufgrund der vielen Vorteile von XML und XSLT, besonders im Zusammenspiel, wird der Hype um XML nicht so schnell verschwinden. Durch die neuen Möglichkeiten, die XSLT bietet, wird der Hype eher zunehmen.

5 Anhang

5.1 Kapitel 2.2 – Strukturen eines XML-Dokuments



Baumstruktur des Beispiels 2.2.1 – das erste XML-Dokument

5.2 Kapitel 3.4 – Ein erstes Beispiel

5.2.1 Das XML-Dokument

```
1  <?xml version="1.0">
2  <Buchliste>
3          <Buch type="t">
4                  <Titel>Wien wartet auf dich!</Titel>
5                  <Autor>
6                          <Vorname>Tom</Vorname>
7                          <Nachname>DeMarco</Nachname>
8                  </Autor>
9                  <ISBN>3-446-16229-1</ISBN>
10         </Buch>
11         <Buch type="g">
12                 <Titel>Einführung in XML</Titel>
13                 <Autor>
14                         <Vorname>Erik</Vorname>
15                         <Nachname>Ray</Nachname>
16                 </Autor>
17                 <ISBN>3-89721-286-2</ISBN>
18         </Buch>
19         <Buch type="g">
20                 <Titel>Mythos Motivation</Titel>
21                 <Autor>
22                         <Vorname>Reinhard</Vorname>
23                         <Nachname>Sprenger</Nachname>
24                 </Autor>
```

```
25                  <ISBN>3-593-37637-7</ISBN>
26          </Buch>
27 </Buchliste>
```

XML-Dokument des Beispiels aus Kapitel 3.4

5.2.2 Die erste Vorlage

```
1  <?xml version="1.0" ?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3          <xsl:output method="html" indent="yes" />
4          <xsl:template match="/">
5                  <html>
6                          <head>
7                                  <title>Bücherliste</title>
8                          </head>
9                          <body>
10                                 <h1>Die Bücherliste</h1>
11                         </body>
12                 </html>
13         </xsl:template>
14 </xsl:stylesheet>
```

Die erste Vorlage aus dem Beispiel des Kapitels 3.4

5.2.3 Wiederholungen

```
1  <?xml version="1.0" ?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3          <xsl:output method="html" indent="yes" />
4          <xsl:template match="/">
5                  <html>
6                          <head>
7                                  <title>Bücherliste</title>
8                          </head>
9                          <body>
10                                 <h1>Die Bücherliste</h1>
11                                 <ul>
12                                 <xsl:for-each select="Buchliste/Buch">
13                                         <li>
14                                                 <xsl:value-of select="Titel" />
15                                                 <xsl:text> - </xsl:text>
16                                                 <xsl:value-of select="Autor/Nachname" />
17                                                 <xsl:text> - </xsl:text>
18                                                 <xsl:value-of select="ISBN" />
19                                         </li>
20                                 </xsl:for-each>
21                                 </ul>
22                         </body>
23                 </html>
24         </xsl:template>
25 </xsl:stylesheet>
```

die Vorlage wurde durch eine Schleife erweitert

5.2.4 Fallunterscheidungen

```
1  <?xml version="1.0" ?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3          <xsl:output method="html" indent="yes" />
4          <xsl:template match="/">
5                  <html>
6                          <head>
7                                  <title>Bücherliste</title>
8                          </head>
9                          <body>
10                                 <h1>Die Bücherliste</h1>
11                                 <ul>
12                                 <xsl:for-each select="Buchliste/Buch">
13                                         <xsl:if test="@type='g'">
14                                         <li>
15                                                 <xsl:value-of select="Titel" />
16                                                 <xsl:text> - gebundene Ausgabe</xsl:text>
17                                         </li>
18                                         <xsl:if>
19                                         <xsl:if test="@type='t'">
20                                         <li>
21                                                 <xsl:value-of select="Titel" />
22                                                 <xsl:text> - Taschenbuch</xsl:text>
23                                         </li>
24                                         <xsl:if>
25                                 </xsl:for-each>
26                                 </ul>
27                         </body>
28                 </html>
29         </xsl:template>
30 </xsl:stylesheet>
```

Einbau von Fallunterscheidungen

6 Literaturverzeichnis

6.1 Bücher

- **Marco Skulschus, Marcus Wiederstein:** „XSLT und XPath für HTML, Text und XML“. (1. Auflage 2005)
- **Henning Behme, Stefan Mintert:** „XML in der Praxis“. (1998)
- **Michael Kay:** „XSLT Programmer's Reference“. (1. Auflage Mai 2000)