XML und Webseitengestaltung

Annett Ehrlich annett.ehrlich@stud.tu-ilmenau.de
Sandra Finger sandra.finger@stud.tu-ilmenau.de
Manuel Löffelholz Manuel.loeffelholz@stud.tu-ilmenau.de

02.Januar 2005

Seminar XML und eCommerce
Dipl.-Medienwiss. Marion Riedle, WS 2004/2005
TU Ilmenau

Inhaltsverzeichnis

1. Zusammenfassung

Wie der Titel dieser Hausarbeit nahe legt, beschäftigt sich dieser Aufsatz mit XML in Bezug auf die Gestaltung von Webseiten. Während in der Einleitung nochmals kurz Licht auf die notwendige Entwicklung von XML vor dem Hintergrund der zunehmenden Entfaltung des World Wide Webs geworfen wird, problematisiert der darauf folgende Abschnitt die Unterschiede und die sich daraus ergebenden Vorteile von XML gegenüber HTML.
Mittelpunkt dieses Aufsatzes stellen die Möglichkeiten dar, die XML bei der Gestaltung von Webseiten bietet. Daher ist auch ein großer Teil dieser Arbeit den Instrumenten gewidmet, die hierbei von Bedeutung sind und Anwendung finden. Nachdem das Prinzip der Cascading Style Sheets kurz erläutert wurde, wird die Entwicklung hin zu XSL beschrieben sowie einige der dazugehörigen Werkzeuge dargestellt. Dabei wird auf XSLT, XPath, XSL:Fo und XPointer eingegangen. Des Weiteren wird dann auch SMIL und XLink thematisiert.
Eine kurze Zusammenfassung und ein Ausblick auf die Zukunft von XML rundet die Arbeit letztendlich ab.

2. Einleitung – Von SGML über HTML hin zu XML

Mit der Entwicklung des World Wide Webs wurde zunehmend ein einfaches und im Gegensatz zu der bisherigen Standard Generalized Markup Language (SGML) weniger komplexes Auszeichnungssystem benötigt, das jeder praktisch schnell erlernen kann. Auf dieser Grundlage wurde Ende der 80erJahre/Anfang der 90er Jahre schließlich die Hyper Text Markup Language (HTML) entwickelt. Im Prinzip ist HTML nichts anderes als ein spezieller Dokumenttyp von SGML, welcher zur „Basis für das aufstrebende World Wide Web“[1] wurde. Mittlerweile kann HTML als „die Programmiersprache des World Wide Webs“[2] bezeichnet werden. Dies ist jedoch nicht ganz exakt, da HTML im Grunde genommen „nur“ eine Seitenbeschreibungs- und Auszeichnungssprache ist, die definiert, wie das Erscheinungsbild eines Textdokumentes aussieht. Der Begriff der Programmiersprache scheint also etwas übertrieben zu sein.

Das World Wide Web entwickelte sich mit den Jahren immer weiter, so dass die Möglichkeiten, die HTML zur Gestaltung von Webseiten bietet, ausgereizt wurden. Es zeichnete sich zunehmend ab, dass HTML den Anforderungen professioneller Webseiten keineswegs mehr gerecht werden konnte. Viele Entwickler von Web-Programmen fühlten sich in ihren Möglichkeiten mehr als nur eingeschränkt. Ihrer Meinung nach fehlten dem Web regelrecht die Möglichkeiten von SGML[3]. Dies führte Ende der 90er Jahre dazu, dass neben Erweiterungen für HTML auch an der Entwicklung einer neuen vereinfachten Variante von SGML gearbeitet wurde. Das Ergebnis ist XML, was eigentlich nichts anderes als eine abgespeckte Version der SGML ist. Während die nützlichsten Teile von SGML erhalten blieben, ließ man die weniger gebräuchlichen und sehr komplizierten Teile weg. Die Ziele, welche mit der Entwicklung von XML verfolgt wurden, lassen sich in 10 Punkten zusammenfassen (Anhang auf Seite 32). Unter dem Aspekt der Nutzung von XML zur Gestaltung von Webseiten sind insbesondere drei Fakten von besonderer Bedeutung. Zum einen sollte XML einfach für das Internet nutzbar

[1] vgl. Dünhölter, Kuno (1998)
[2] vgl. Kobert, Thomas (1999)
[3] vgl. Dünhölter, Kuno (1998)

sein und zum anderen sollte XML eine Vielzahl von Anwendungen unterstützen sowie kompatibel zu SGML sein.

3. Vorteile von XML gegenüber HTML

3.1 Metasprache bietet höhere Flexibilität

XML bietet gegenüber HTML vielerlei Vorteile, auch was die Möglichkeiten beim Webdesign angeht. Der grundlegendste Vorteil beruht dabei darauf, dass XML eine Metasprache ist und dadurch höhere Flexibilität bietet.

Das Problem bei HTML ist, dass dem Nutzer eine bestimmte Anzahl an vorgegebenen Tags zur Verfügung steht, egal ob er diese nun benötigt oder nicht. So beispielsweise gibt es Tags zur Darstellung chemischer Formeln oder auch Musiknoten, die eigentlich nur für eine Minderheit aller Webseiten benötigt wird, aber so allgemein gehalten sind, dass sie von jedem verwendet werden können[4].

Durch die Verwendung von XML ist die Kreativität nicht mehr an eine limitierte Anzahl von Tags gebunden. Jeder kann seine eigene Auszeichnungssprache entwickeln, die exakt auf die spezifische Problemstellung und Bedürfnisse abgestimmt ist[5]. Wenn ein Befehl fehlen sollte, dann entwirft der Nutzer sich ihn einfach selbst. Unbrauchbare Tags, die man eh nicht benötigt, fallen hingegen weg.

Des Weiteren ermöglicht XML die Definition von Tags jeder nur denkbaren Art. Während bei HTML Tags „nur“ das Erscheinungsbild eines Textdokumentes definieren, erlaubt XML „Tags, die Daten oder das Verhältnis von Daten zueinander beschreiben“[6]. Dies ist darauf zurückzuführen, dass XML-Tags weniger die Formatierung als vielmehr den Inhalt beschreiben und somit also der Strukturierung und Abgrenzung von Daten dienen. Sie sind daher auch als Metadaten charakterisierbar, was die Grundlage für den zweiten großen Vorteil von XML gegenüber HTML liefert.

[4] vgl. Dobnig, Mario (2000)
[5] vgl. Dobnig, Mario (2000)
[6] vgl. Dobnig, Mario (2000)

3.2 Metadaten erhöhen Informationsgehalt

Durch Metadaten wird der Informationsgehalt bei XML erhöht.
Bisher war die Suche nach Informationen im World Wide Web recht aufwendig. Im Prinzip wurde jede Seite nach jedem einzelnen Suchbegriff durchforstet. Die Suche nach Wörtern, die in einer bestimmten Beziehung zueinander stehen, war damit nicht möglich.
XML scheint in dieser Beziehung einen großen Schritt für die Zukunft zu machen. Da die Tags den Inhalt der Daten beschreiben, erklärt sich XML wie von selbst. Somit werden die Informationen nicht nur für Außenstehende leichter verständlich, sondern sind auch wesentlich gezielter von Suchmaschinen auswertbar. Vielleicht wird dadurch endlich erreicht, dass Suchmaschinen tatsächlich auch das Ergebnis liefern, wonach gesucht wurde[7].

3.3 Trennung von Struktur und Layout

Der dritte Vorteil von XML gegenüber HTML beruht auf der Trennung von Struktur und Layout (Anhang auf Seite 32).
Bei HTML ist die Textformatierung mit den Daten vermischt. Dies stellt sich spätestens dann als problematisch heraus, wenn nun eine Seite geändert werden soll. Egal wie groß oder klein die Veränderung auch sein mag, müssen höchstwahrscheinlich alle Seiten durcharbeitet werden, was natürlich mit viel Aufwand und Zeit verbunden ist.
XML vereinfacht dieses Verfahren durch die Trennung von Daten und Darstellung in unterschiedlichen Dokumenten. Während das XML-Dokument an sich vergleichbar mit Datenbankinhalten ist, wo „nur“ die Daten strukturiert abgelegt sind, ist ein zusätzliches „Stylesheet“ notwendig, um das Layout des Dokuments festzulegen. Somit ist einerseits gewährleistet, dass eine Datenbasis für unterschiedliche Anwendungen genutzt werden kann (beispielsweise können Wetterdaten sowohl als Grafik als auch als Tabelle ausgegeben werden)[8] und andererseits sind Änderungen im Dokument mit weniger Aufwand verbunden. Je

[7] vgl. Dobnig, Mario (2000)
[8] vgl. Wikimedia Foundation Inc. (2004)

nachdem, ob nun die Daten oder das Layout verändert werden soll, kann auf das XML-Dokument oder das Stylesheet direkt zugegriffen werden.

4. Cascading Stylesheets CSS

Wie schon im einführenden Kapitel angesprochen wurde, handelt es sich bei XML um eine semantische Markup Language. Diese liefert keine Informationen über die Darstellung der einzelnen Elemente, sondern lediglich Angaben zu ihrer logischen Bedeutung und wird deshalb auch als „logische Auszeichnungssprache bezeichnet“

Um die im XML- Dokument enthaltenden Daten in einem Browser darstellen zu können, ist eine Transformation nötig. Wie die Abbildung 1 zeigt, ist solch eine Transformation durch Style Sheets zu generieren.

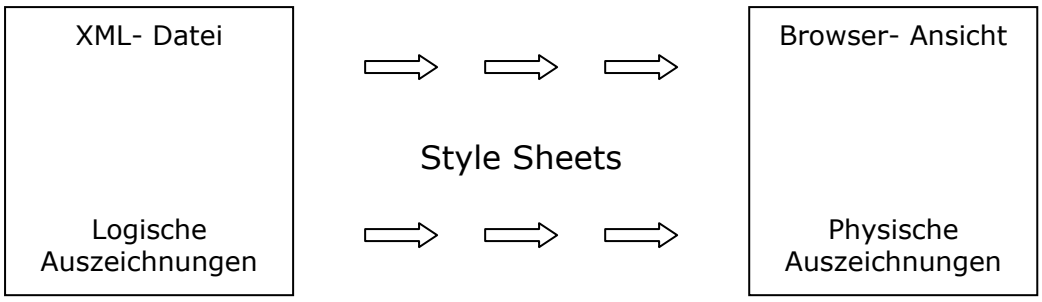


Abbildung 1

Um eine solche Browser- Ausgabe zu erreichen hat man 2 Möglichkeiten. Die erste Variante ist die Ausgabe mittels Cascading- Stylesheets. Cascading Stylesheets enthalten einen Konstruktionsplan für die Darstellung der Quelldatei. Sie erlauben es, den im XML- Dokument definierten Tags Gestaltungsinformationen zuzuordnen. CSS dienen folglich als Vorlage zur Umwandlung der logischen Auszeichnungen in die physischen Auszeichnungen.

Die zweite Möglichkeit ist die Ausgabe mittels XSL-Datei mit der sich das … Kapitel beschäftigt. Eine andere Darstellungsmöglichkeit, auf die hier aber nicht weiter eingegangen wird, ist die Formatierung mittels XHTML- Tags. Hier werden direkt in der XML- Datei XHTML- Tags eingearbeitet, was einen

logischen Bruch zu dem eigentlichen Ziel von XML(Trennung von Struktur und Inhalt) darstellt.
Während die ursprünglichen Funktionen von CSS auf HTML beschränkt waren, wurden in der CSS- Version 2.0 einige Veränderungen im Bezug auf XML einbezogen. Mit CSS ist es jedoch nicht möglich einzelne Elemente in eine komplexere Struktur von Objekten zu integrieren und eine Interaktivität bzw. dynamische Gestaltung von Webseiten zu erreichen. Beispielsweise können mit CSS keine Inhaltsverzeichnisse, Seiten- und Kapitelnummerierungen oder feste Kopf- und Fußzeilen automatisch erstellt werden.
Die Funktionsvielfalt von CSS beschränkt sich somit auf eingeschränkte Veränderungen an einzelnen Elementen.
Wenn aufwendigere Formatierungen zu realisieren sind, dann sollte man die extra für XML entwickelte „Extensible Stylesheet Language“ nutzen.

4.1 Stylesheets referenzieren

Die CSS-Stylesheet-Datei wird direkt im XML- Dokument referenziert. Nach der Referenz werden die Tags mit den jeweiligen Inhalten erstellt, die mittels Stylesheet- Datei formatiert werden.

<?xml:stylesheet *{hierdurch wird dem Browser mitgeteilt, dass eine Stylesheet-Definition folgt}*

type="text/css" *{dadurch wird dem Browser mitgeteilt, dass dieses Dokument lediglich Textelemente enthält}*

href="adresse.css" *{hier wird auf die CSS-Datei verwiesen}***?>**

4.2 Beispiel: XML- Datei mit CSS- Stylesheet und Browserausgabe

Ein Beispiel für eine Browserausgabe einer XML-Datei mittels Cascading Stylesheets befindet sich im Anhang auf Seite 33.

5. XSL Extensible Stylesheet Language

Die erweiterbare Stylesheet- Sprache XSL erwächst aus dem Sprachraum der XML- Familie und stellt eine der komlpiziertesten Spezifikationen dar. XSL teilt sich in zwei Bereiche auf: XSLT, dass für Transformationen benutzt wird und XSL-FO (Formatting Objekts), mit dem wie der Name schon ausdrückt Objekte formatiert werden.

Nachdem die Formatierung von XML- Dokumenten mittels CSS schon erläutert wurde, wird nun auf die Darstellung und Veränderung von XML- Dokumenten mit der Extensible Stylesheet Language(XSL) eingegangen. Sie ist eleganter und den Ansprüchen von XML besser gewachsen, da sie „größere Möglichkeiten bezüglich der Formatierung und des Funktionsumfanges zur Verfügung stellt.“[9]

XSL unterstützt die meisten Funktionen, die auch aus CSS bekannt sind und bietet außerdem noch weitere Möglichkeiten der Auszeichnung.

Über die reine Formatierung hinaus bietet XSL eine Sprache, mit deren Hilfe XML- Instanzen und Formatting Objekts transformiert werden können. XSL macht sich die Baumstruktur eines XML- Dokumentes zunutze indem es Formatierungsregeln auf einen Quellbaum anwendet.[10] Mit XSLT können Transformationsregeln auf eine Dokumentenquelle angewendet und durch Änderung der Baumstruktur ein neues Dokument erzeugt werden. Weiterhin macht es XSLT möglich mehrere Dokumente in ein einziges Dokument zu überführen bzw. aus einer XML- Datei mehrere Dokumente zu erzeugen. Mittels XSL- Definition können somit XML- Dokumente in andere Formate konvertiert werden. Die häufigsten Formate sind hierbei: HTML, RTF oder PDF.

Der Inhalt der generierten HTML-, RTF- oder PDF- Dokumente entstammt der XML- Datei und die visuelle Umsetzung ist im zugehörigen XSL- Stylesheet festgelegt.

Die folgende Abbildung 2 soll die eben erwähnten Möglichkeiten, die XSL bietet, visualisieren.

[9] Harms, Koch, Kürten 2000, S.547
[10] vgl. Eckstein R. & Casabianca M. 2002, S. 44

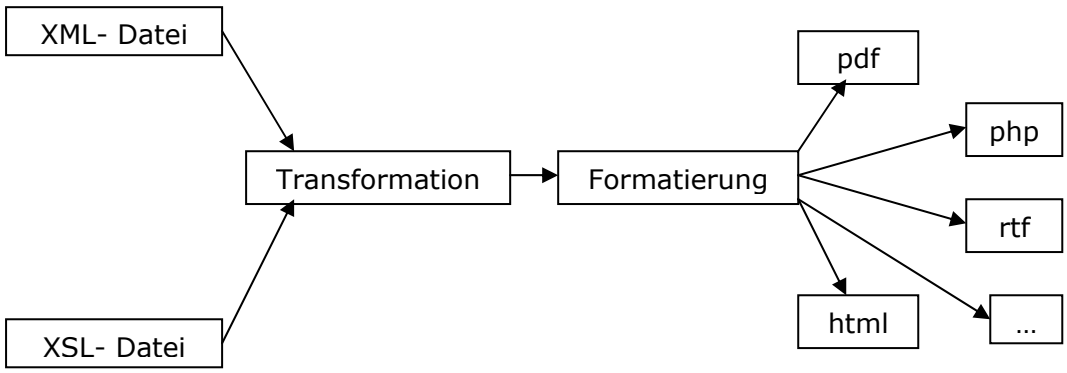


Abbildung 2

5.1 Formatierung von Elementen

Mit der Extensible Stylesheet Language XSL bestehen unter anderem folgende Formatierungsmöglichkeiten:

- Schriftarten / Schriftgrößen
- Farben
- Vertikale und horizontale Ausrichtungen
- Gliederung (Überschriften)
- Tabellen
- Behandlung von Leerräumen (Leerzeichen und -zeilen)

5.2 Die zwei Hauptaufgaben von XSL

Die XSL hat vor allem zwei Hauptaufgaben. Die erste liegt in der Bildung einer Sprache mit der sich XML- Dokumente in andere Formate konvertieren lassen (Beispielsweise HTML oder PDF). Die zweite liegt darin, ein Vokabular zur Verfügung zu stellen, das den semantischen Tags bestimmte Formatierungen zuweist (ähnlich der bestehenden CSS).

5.3 Einführung in die Programmierung von XSL

Im Folgenden werden einige Grundregeln für die XSL-Programmierung dargestellt. Es wird das Grundgerüst erläutert und auf die Konstruktionsregeln eingegangen. Anschließend wird die Verwendung von Templates dargelegt und anhand eines Beispiels verdeutlicht.

5.3.1 Grundgerüst einer XSL-Datei:

```
<xsl>
<rule>
<!-- Konstruktionsregel 1 -->
</rule>
<rule>
<!-- Konstruktionsregel 2 -->
</rule>
</xsl>
```

XSL- Anweisungen werden in einem externen Dokument mit der Endung ».xsl« abgelegt. Diese Datei enthält die physischen Übersetzungen zu den im XML-Dokument definierten Befehlen. Sie muss außerdem in der XML- Datei korrekt referenziert werden um die gewünschten physischen Auszeichnungen darstellen zu können.

5.3.2 Aufbau der Konstruktionsregel

Eine einzelne Stilanweisung oder auch Konstruktionsregel gliedert sich in zwei zusammenhängende Teile:
Das Muster (pattern) stellt den ersten Teil dar.
Durch das dieses wird festgelegt auf welchen (selbstdefinierten) XML-Befehl sich die dann folgende Stilanweisung (*action*) bezieht. Das Muster ist das Auswahlkriterium, wann die definierte Ausgabeform auf den Inhalt eines Markups umzusetzen ist.

Die Aktion (action) ist der zweite Teil der Konstruktionsregel.
Wurde das angegebene Muster im XML-Dokument erkannt, dann folgt die Umwandlung des betreffenden Elements in die angegebene Ausgabeform. Eine »action«-Anweisung kann neben passiven Elementen zur reinen Stilgestaltung auch dynamische Anweisungen, beispielsweise den Aufruf eines JavaScripts oder anderer Skriptsprachen enthalten.

5.3.3 XSLT- Stylesheet- Struktur

Es gibt ein paar einfache Regeln beim programmieren von XSL- Stylesheets, die auf die Einhaltung einer allgemeinen Reihenfolge hin zielt. Übergreifend ist vorab noch zu nennen, dass alle XSL- Stylesheets wohlgeformte XML- Dokumente sein müssen. Sie müssen somit den Regeln der „Wohlgeformtheit“ entsprechen.[11]
Erstens müssen alle XSL- Stylesheets mit dem Start-Tag des XSL-Wurzelelements beginnen.
<xsl:stylesheet version=“1.0“
Und am Ende des Dokuments mit dem entsprechenden End-Tag geschlossen werden.
<xsl:stylesheet/>
Zweitens muss jedes <XSL>- Element den Namensraum verwenden, der durch xmlns- Deklaration im <stylesheet>- Element(Start-Tag) spezifiziert ist.
<xsl:stylesheet version=“1.0“
xmlns:xsl=http://www.w3.org/1999/XSL/Transform>
Nach dem Wurzelelement können zusätzlich noch externe Stylesheets mit dem Element *<xsl:import>* importiert werden. Somit wird die Aufteilung größerer Projekte in einzelne Module möglich und die Wiederverwendung von fertigen XSL-Dokumenten wird dadurch erleichtert.

5.3.4 Templates

Durch die Anwendung von Templates wird ein XML-Dokument für einen gegebenen Knotentyp transformiert. Das Template- Element hat folgendes Aussehen:
<xsl:template match=“Knotentyp“>
…
</xsl:template>
Bei Knotentyp wird der Knotentyp angegeben, der verarbeitet werden soll. In unserem Beispiel im Anhang auf Seite 34 wird mit der Zeile

[11] zur Vertiefung siehe: Eckstein R. & Casabianca M. 2002, S. 18

<xsl:template match="/"> der Knoten „/“, der für das Wurzelelement steht, in der nachfolgenden Spezifikation in HTML transformiert.
Es wird mittels html- Tags eine Tabelle generiert die eine Rahmenstärke von „4pt“ und eine Rahmenfarbe „gelb“ besitzt. Anschließend werden in einzelne Zellen mit vordefinierter Schriftgröße und Farbe die Wörter Vorname, Nachname, Straße, Hausnummer, Postleitzahl und Ort geschrieben. Hiernach wird durch das Tag *<xsl:apply-templates/>* rekursiv alle anderen Templates dieses Stylesheets auf das Wurzelelement „/“ an. Das heißt, dass auch das später definierte Template „Dokument“ auf das Wurzelelement Anwendung findet.
Anschließend wird durch den Aufruf *<xsl:template match="Dokument">* ein Template für das Tag „Dokument“ beschrieben. Durch das folgende Schleifen-Element *<xsl:for-each select="Adresse">* werden alle <Adresse>- Elemente in nachfolgender weise bearbeitet. Es werden jeweils Tabellenzellen erstellt und durch die nachfolgenden Anweisungen *<td><xsl:value-of select="Vorname"/></td> <td><xsl:value-of select="Nachname"/></td>…* werden die Inhalte des in Anführungsstrichen stehenden Tags in die Zellen geschrieben.
Im Anhang auf Seite 34 Kann man die Browserausgabe der Datei Adresse.xml durch die Stylesheet-Datei Adresse.xsl betrachten.

Zusammenfassung der verwendeten Tags:

<?xml:stylesheet *{hierdurch wird dem Browser mitgeteilt, dass eine Stylesheet- Definition folgt}*
type="text/xsl" *{dadurch wird dem Browser mitgeteilt, dass dieses Dokument lediglich Textelemente enthält}*
href="style.xsl" *{hier wird auf die XSL-Datei verwiesen}***?>**

xsl:apply-templates *{Templates werden angewendet}*

xsl:for-each *{Tag erlaubt, alle Tags eines bestimmten Types im aktiven Template **auszugeben** und zu **bearbeiten**. Daher ist es z.B. für Inhaltsverzeichnisse und Stichwortlisten prädestiniert.}*

<xsl:for-each select="//Adresse"> {findet alle (// entspricht an jeder Position des

Dokuments) „Adressen“ des Dokuments und gibt sie mit Namen aus → interessanter Nebeneffekt = Element hinter “select” wird zum aktiven Knoten

<xsl:value-of select="Vorname"/> *{Zwischen den Anführungszeichen steht der Wert der ausgegeben wird}*

5.3.5 Stylesheets referenzieren

Das Stylesheet-Dokument wird durch folgende Zeile direkt im XML-Dokument referenziert.

<?xml-stylesheet type="text/xsl" href="sort.xsl"?>

In dem Sonderfall, dass gleichzeitig ein XSL- und CSS-Stylesheet im XML-Dokument referenziert ist, wird automatisch das XSL-Stylesheet bevorzugt behandelt.

6. Werkzeuge von XSL

6.1 XSLT

6.1.1 Transformationen mit XSLT

Der XSLT-Teil von XSL hat die Aufgabe, Elemente einer XML-basierten Sprache in eine andere XML-gerechte Sprache zu transformieren. Beispielsweise werden Elemente aus einer eigenen XML-Datei, wie vorname und zuname, in Auszeichnungs-Konstrukte einer anderen Sprache transformiert, um damit eine formatierte Ausgabe der Elemente zu erzeugen. Um XML-Daten in HTML zu transformieren muss eine Verbindung zwischen Elementen und Attributen der XML-Daten und bestimmten HTML-Konstrukten hergestellt werden.

Zum Beispiel wird im XSLT-Stylesheet angeben, dass ein Element namens vorname in den HTML-Code <td>[...]</td> umgesetzt werden soll. Aus einer Notation, wie <vorname>Stefan</vorname> wird dann daraus beim Transformieren als Ergebnis das HTML-Konstrukt <td>Stefan</td> erzeugt. Bei der Transformation wird aus einem Quellbaum ein Ergebnisbaum erzeugt. Dies ist möglich, weil sich alle XML-basierten Daten als Baumstruktur darstellen lassen.

Es ist oft nötig, ganz gezielt auf einzelne Bestandteile (Knoten) der XML-Daten im „Baum“ zuzugreifen, um diese beim Ezeugen des Ergebnisbaums ebenfalls an ganz bestimmten Stellen unterzubringen. Dafür benutzt XSLT eine Syntax, die in einer eigenen Sprache definiert wird: in XPath. XPath (siehe Kapitel 6.2) beschreibt, wie man in XML-gerechten Datenstrukturen auf beliebige Strukturbestandteile zugreifen kann.

6.1.2 XSLT-Elemente

Es gibt sehr viele Element in XSLT: z.B. xsl:apply-import, xsl:for-each, xsl:apply-templates, xsl:if, xsl:attribute, xsl:import, xsl:call-template, xsl:key, xsl:choose, xsl:message, xsl:comment, xsl:number, xsl:copy, xsl:output, xsl:element, xsl:sort, xsl:fallback, xsl:stylesheet, xsl:strip-space, xsl:text, usw.

Im Nachfolgenden werden nun 3 davon als Beispiele herausgenommen und näher erläutert.

6.1.2.1 Eine Auswahl treffen mit xsl:choose

Bildet den Rahmen für eine Reihe von Abfragen, die mit xsl:when und xsl:otherwise durchgeführt werden. Die Reihe kann aus beliebig vielen xsl:when-Abfragen und einer abschließenden xsl:otherwise-Anweisung bestehen. Ausgewählt wird diejenige Abfrage, deren Bedingung als erste zutrifft.

xsl:choose hat keine Attribute. Kann innerhalb von xsl:template vorkommen.

Beispieldatei *choose.xml* siehe Anhang Seite 35.

Im Beispiel-Stylesheet wird mit <xsl:template match="postleitzahl"> das Template definiert, das die HTML-Ausgabe der Zahlen steuert. Mit <xsl:value-of select="." /> wird der Wert der Zahl ausgegeben. Anschließend wird eine Variable namens *wert* definiert, die sich den Wert, also den Inhalt zwischen <zahl> und </zahl>, merkt. Mit <xsl:choose> wird sodann eine Abfragereihe eingeleitet. Sie enthält eine xsl:when-Abfrage und eine xsl:otherwise-Anweisung. Bei der xsl:when-Abfrage wird abgefragt, ob der Wert der Variablen *wert* kleiner als 10000 ist. Wenn ja, wird hinter die bereits ausgegebene Zahl der Text (unvollständige Postleitzahl) ausgegeben, andernfalls (otherwise) kein Text. Für die Textausgabe wird xsl:text benutzt.

Zur bildlichen Darstellung wurde eine der Postleitzahlen (aus Adresse.xml) manipuliert und das ganze im Internet Explorer ausgegeben:

Die Postleitzahlen lauten:

98693
986 (unvollständige Postleitzahl)
98693

Also erscheint nun hinter einer Zahl der Text, während bei den anderen beiden nichts dahinter steht.

6.1.2.2. Einen Kommentar setzten mit xsl:comment

Schließt Ausgaben im Ergebnisbaum in XML- (auch HTML)-gerechte Kommentare ein, sodass der Inhalt nicht ausgegeben wird. xsl:comment
hat keine Attribute. Kann ebenfalls innerhalb von xsl:template vorkommen.
Beispielauszug aus einem XSL-Stylesheet siehe Anhang Seite 36.
Im Beispiel-Stylesheet wird bei der HTML-Ausgabe unter anderem ein JavaScript-Bereich notiert. Der Inhalt eines solchen Bereichs sollte, um den Inhalt für nicht-javascript-fähige Browser unzugänglich machen, auskommentiert werden. Im Beispiel wird dies mit Hilfe der xsl:comment-Anweisung realisiert.
Im Fenster des Internet Explorers öffnet sich ein Pop-Up in dem steht: „ Und schon erscheint ein Kommentar!“

6.1.2.3. Elemente nach Inhalt sortieren mit xsl:sort

Sortiert werden die Knoten aus einer Reihenfolge nach ihren Inhalten. So lassen sich beispielsweise alle untergeordneten Elemente listenpunkt eines Elements namens liste bei der Transformation alphabetisch oder numerisch sortieren.
Hat folgende Attribute:
caseorder = bestimmt, ob in der Sortierung Großbuchstaben vor Kleinbuchstaben kommen oder umgekehrt. *Upper-first* bedeutet Großbuchstaben zuerst und *lower-first* Kleinbuchstaben zuerst.
datatype = bestimmt, ob die Sortierung alphabetisch oder numerisch erfolgen soll. Bei alphabetischer Sortierung kommt z.B. 9 nach 10, bei numerischer 10 nach 9. *Text* bedeutet alphabetische Sortierung und *number* numerische Sortierung.
lang = gibt das Land an, nach dessen Sprachkonventionen die Sortierung erfolgen soll. Beispielsweise wird ein ä im Deutschen (lang=“de“) anders im Alphabet eingeordnet als im Schwedischen (lang=“se“).

order = bestimmt, ob die Sortierung aufsteigend, also von A nach Z, oder absteigend, also von Z nach A, erfolgen soll. Ascending bedeutet eine aufsteigende Sortierung und descending eine Absteigende.
select = gibt an, was sortiert werden soll.
Kann auch innerhalb von xsl:apply-templates oder xsl:for-each vorkommen.
Beispieldatei *sort.xsl* siehe Anhang Seite 36.

Im Beispiel wird bei der HTML-Ausgabe diese Liste (Adresse.xml) nach Hausnummern absteigend sortiert. Dazu wird im Beispiel-Stylesheet innerhalb der xsl:for-each-Anweisung eine xsl:sort-Anweisung notiert. Die Anweisung wirkt an dieser Stelle notiert wie ein Filter, der die Abarbeitung der einzelnen Elemente vom Typ adresse beeinflusst. Die Sortierreihenfolge soll absteigend (descending) sein. Da die Hausnummern numerisch interpretiert werden sollen, ist noch die Angabe data-type="number" notiert. Anschließend werden einfach die aktuellen Werte der Elemente Vorname und Hausnummer in die HTML-Tabelle geschrieben. Die Ausgabe im Internet Explorer sieht dann wie folgt aus:

Annett	9
Sandra	5
Manuel	3

6.2 XPATH

XPath ist auch ein Sprachprojekt des W3-Konsortiums, allerdings ist XPath lediglich eine Art Hilfssprache, die erforderlich ist, damit XSLT seine Aufgaben wahrnehmen kann.[12]
Xpath hat dabei **drei** wichtige Aufgaben:

- Adressierung von Daten
- Definition logischer Ausdrücke
- Zusätzliche Funktionen

6.2.1. Adressierung von Daten

Beim Übersetzen einer XML-Dokumentstruktur in eine andere XML-Dokumentstruktur, also etwa beim Übersetzen eines Dokuments mit eigener

[12] vgl. SELFHTML e.V. (2001)

XML-DTD in ein HTML-Dokument, ist es wichtig, jeden Bestandteil der Datenstruktur genau ansprechen zu können. Wenn Sie im (XML)-Ausgangsbaum beispielsweise stehen haben: <augen farbe="graublau">, und Sie möchten im Ergebnisbaum daraus erzeugen: <th>Augenfarbe</th><td>graublau</td>, dann müssen Sie auf das Attribut *farbe* des Elements *augen* zugreifen. Angenommen weiter, das Element *augen* ist sowohl ein Kindelement eines Elements namens *person*, als auch ein Kindelement eines Elements namens *phantombilddaten* in einem anderen Zweig des geichen XML-Dokuments. Daraus kann man erahnen, dass für die eindeutige Adressierung von Dokumentbestandteilen eine brauchbare Adressierungs-Syntax benötigt wird. Typische Fälle, in denen die XPath-Adressierung zum Einsatz kommt, sind z.B. das select-Attribut der xsl:value-of-Anweisung oder das match-Attribut der xsl:template-Anweisung.

Die Baumstruktur von XML-Dokumenten benötigt eine genaue Adressierung. Ebenso, wie man bei der Dateistruktur auf einem Datenträger von Verzeichnissen oder Ordnern, Dateien und Pfadangaben für Dateien spricht, braucht man für die Adressierung von Bestandteilen eines XML-Dokuments eine geeignete Terminologie. Bei XPath gibt es dafür verschiedene Knotentypen, Achsen und Pfade. Entscheidend sind die englischen Originalbegriffe.[13]

Als Knoten-Typen werden die unterschiedlichen Teile eines XML-Dokuments wie Elemente, Attribute und Kommentare repräsentiert

Achsen beschreiben die unterschiedlichen Beziehungen der einzelnen Knoten zueinander. Es bestehen Abhängigkeiten oder die Knoten liegen auf einer gemeinsamen Ebene. Es werden verschiedene Begriffe aus der Verwandtschaftsterminologie genutzt, die als Achsen bezeichnet werden. XPath unterscheidet 13 Achsen, die bei der ausführlichen Adressierung verwendet werden.

In XPath gibt es 2 syntaktische Mittel, um den Pfad zu einem oder mehreren Knoten zu notieren. Ein einfacher Schrägstrich (/) ist das Pfad-Trennzeichen für tatsächliche Namen von Knoten, d.h. in einer Struktur wie <a><c>...</c><c>...</c> lässt sich das c-Element als Pfadangabe in der Form a/b/c darstellen. Das Zweite ist der doppelte Doppelpunkt (::). Er ist das

[13] vgl. Knobloch, M. & Kopp, M. (2001), S. 125ff

Pfad-Trennzeichen für Knotenangaben mit Achsen, d.h. in einer Struktur wie <a><c>...</c><c>...</c> lässt sich das c-Element als Pfadangabe z.B. in der Form a/b/child::c darstellen.

Es gibt eine ausführliche und eine verkürzte Notation. Die ausführliche enthält die vollen Achsenbezeichnungen, z.B. /child::augen/attribute::farbe. In der verkürzten Notation wird auf Achsenbezeichnungen verzichtet, z.B. /augen/@farbe. Am meisten wird die verkürzte Notation angewandt, deswegen wird im Folgenden nur diese verwendet.

6.2.1.1 Adressierung der Dokumentwurzel:

```
<xsl:template match="/">
<html>
<head>
        <title>
                Beispiel
        </title>
</head> <body>
        <xsl:apply-templates />
</body> </html>
</xsl:template>
```

Der Wurzelknoten eines XML-Dokuments wird mit einem einzelnen Schrägstrich (/) adressiert. Im Beispiel wird mit xsl:template ein Template für die XML-Dokumentwurzel definiert. Im match-Attribut wird die Dokumentwurzel adressiert.

6.2.1.2 Adressierung mit absoluten Pfadangaben:

```
<xsl:template match="/adresse/vorname">
<b>
<xsl:value-of select="/adresse/vorname">
</b>
</xsl:template>
```

beginnen mit einem Schrägstrich (/), der die Dokumentwurzel repräsentiert. Weitere Hierarchie-Ebenen werden durch weitere Schrägstriche markiert. Im Bsp. wird mit der xsl:template-Anweisung ein Template für ein Element namens vorname definiert, das ein Kindelement des Elements adresse ist, das wieder ein Kindelement der Dokumentwurzel ist

Wenn man mit relativen Pfadangaben adressiert, geht man von einem aktuellen Standpunkt aus, d.h. man muss nicht vom Wurzelelement aus adressieren, sondern von einem Punkt aus, der tiefer im Dokument drin ist.

6.2.1.3 Adressierung von Attributen:

```
<xsl:template match="augen/@farbe">
        <tr> <td>
        <b>
                <xsl:text>Augenfarbe:</xsl:text>
        </b>
        </td> <td>
                <xsl:value-of select="." />
        </td> </tr>
</xsl:template>
```

Das Beispiel nimmt an, dass es eine XML-Notation <augen farbe="..."> gibt. Für das Attribut wird ein eigenes Template definiert. Dabei wird im HTML-Ergebnisbaum eine zweispaltige Tabellenzeile erzeugt, in der links das statische Wort Augenfarbe: steht, während in der rechten Spalte der Wert des Attributs farbe= ausgegeben wird. Das match-Attribut der Template-Definition wird so adressiert, dass das Template von einem Elternelement von augen aus - z.B. von person - aufgerufen werden kann. Der Pfad lautet dann in der verkürzten, Notation augen/@farbe.

6.2.2. Definition logischer Ausdrücke

Damit sind logische Ausdrücke gemeint, die vom XSLT-Prozessor mit verarbeitet werden, und die zur Funktionalität bestimmter XSLT-Elemente wichtig sind. Logische Ausdrücke enthalten Operatoren und können damit z.B. zwei Werte vergleichen. Ähnlich wie die Adressierung von Daten kommt die Definition von Ausdrücken in bestimmten Wertzuweisungen an Attribute von XSLT-Elementen vor. Ein typisches Beispiel ist das *test*-Attribut der xsl:if-Anweisung. Darin wird eine Bedingung formuliert, die wahr oder falsch sein kann. Nur wenn sie wahr ist, werden untergeordnete Anweisungen ausgeführt. Eine Bedingung wie *test="alter >= 18"* (hat das Element alter einen Wert größer gleich 18?) ist so ein typischer logischer Ausdruck. Die Syntax dabei basiert auf XPath.

XPath-Operatoren

Sie erlauben es, Werte zu vergleichen, Berechnungen durchzuführen, oder Ausdrücke zu bewerten. XPath Operatoren können in Attributen

bestimmter XSLT-Elemente vorkommen, helfen Bedingungen zu formulieren, Elementpositionen zu berechnen und anderes mehr.

Beispiele für solche Operatoren wären: + Addition; - Subtraktion; *Multiplikation; = gleich (Vergleich zweier Werte); != ungleich (Vergleich zweier Werte); < kleiner als; < größer als; <= kleiner oder gleich; *and* logisches und; *or* logische Oder; *div* Fließkommateilung; *mod* Rest der Fließkommateilung.

6.2.3. Zusätzliche Funktionen

Seinen großen Leistungsumfang erhält XSLT nicht zuletzt durch ein Arsenal an zusätzlichen Funktionen, die bestimmte Aufgaben wahrnehmen. So gibt es z.B. eine Funktion *position(),* die beim Abarbeiten mehrerer Elemente in einer Reihe zurückliefert, das wievielte Element gerade bearbeitet wird. Diese Information lässt sich beispielsweise dazu nutzen, um die Elemente im Ergebnisbaum zu nummerieren. Eine andere typische Funktion namens *starts-with()* überprüft, ob eine Zeichenkette mit einer bestimmten Teilzeichenkette beginnt. Wenn diese Funktion etwa als Bedingung in einer xsl:if-Anweisung notiert wird, lässt sich das Erzeugen des Ergebnisbaums von der Abfrage beeinflussen, ob der Wert eines Elements mit einem bestimmten Zeichen oder einer bestimmten Zeichenfolge beginnt. So könnte eine Bedingung wie *test="starts-with(plz,'8')"* zum Beispiel abfragen, ob der Wert einer Postleitzahl mit 8 beginnt, und nur Elemente in den Ergebnisbaum übernehmen, bei denen dies der Fall ist. Nachfolgend werden nun beispielhaft 2 Funktionen näher erläutert.[14]

6.2.3.1. Positionsnummer des aktuellen Knotens ermitteln mit position()

```
<xsl: variable name="anzahl" select="count(//Adresse)"/>
<xsl:template match="Dokument">
        <xsl:for-each select="Adresse">
        <p>
                <xsl:value-of select="position()" />
                <xsl:text>. Von </xsl:text>
                <xsl:value-of select=$anzahl"/>
                <xsl:text>: </xsl:text>
                <xsl:value-of select="." />
        </p>
        </xsl:for-each>
</xsl:template>
```

[14] vgl. SELFHTML e.V. (2001)

Es gibt eine Liste mit mehreren Adressen. Zunächst wird mit *xsl:variable* eine Variable Anzahl definiert, die die Anzahl der Listeneinträge speichert. In der *for-each* Schleife werden alle Listeneinträge abgearbeitet und vor dem Text des Listeneintrags die aktuelle Positionsnummer mit *position()* angegeben, und dahinter die Gesamtzahl, die in der Variablen *anzahl* gespeichert sind.

Zum Beispiel so: 1. von 3.: Text des ersten Listeneintrags (die erste Adresse)

2. von 3.: Text des zweiten Listeneintrags (die zweite Adresse)

6.2.3.2. Anzahl Zeichen einer Zeichenkette ermitteln mit string-length()

Ermittelt wie lang eine Zeichenkette ist. Und liefert die Anzahl der Zeichen zurück.

```
<xsl:template match=”mitteilungstext”>
<xsl:choose>
        <xsl:when test=”string-length() &lt; 20”>
        <p><b>Fehler: zu geringe Mitteilungsmenge!</b></p>
        </xsl:when>
        <xsl:otherwise>
                <p><xsl:value-of-select=”.” /></p>
        </otherwise>
 </xsl:choose>
</xsl:template>
```

Es wird ein Template für ein Element namens *mitteilungstext* definiert. Der Inhalt des Elements wird nur übernommen, wenn er mindestens 20 Zeichen besitzt. Dazu wird mit xsl:choose eine logische Verzweigung realisiert. Im xsl:when Zweig wird abgefragt, ob der Elementinhalt kleiner als 20 Zeichen ist. Dazu wird die Funktion string-length() angewendet. Da kein Argument übergeben wird, bezieht sie sich automatisch auf den Inhalt des aktuellen Elements. Wenn weniger als 20 Zeichen ermittelt werden, wird eine Fehlermeldung in die Ausgabe geschrieben. Andernfalls (xsl:otherwise) wird der Inhalt vom *mitteilungstext* in den Ergebnisbaum geschrieben.

6.3 XSL:FO

Die Formatting Objects (FO) sind die XML-Elemente, die den Sprachumfang der Formatierungssprache XSL darstellen. Sie können auf jeden Bereich angewandt werden, einschließlich Video und Printmedien. Diese Sprache erlaubt es, Angaben zum Layout, Typographie, Linien, Blöcken, Fußnoten usw. zu machen. Ein Formatierungsprogramm verarbeitet das FO-Dokument in ein konsumierbares

Dokument (wie z.B. PDF).Ein und dasselbe FO-Dokument kann in verschiedene Dateiformate überführt werden. Das FO-Dokument lässt sich in 2 Abschnitte einteilen:

1. <fo:layout-master-set> - es werden Seitentypen für das aktuelle Dokument definiert und die Abfolge dieser Seitentypen beschrieben.
2. <fo:page-sequence> - ein Folge von diesen Elementen stellt die „Beschreibung der Seitenbefüllung“ dar.

Es kann lediglich ein layout-master-set, aber mehrere page-sequences in einem Dokument enthalten sein. [15]

Fragment eines FO-Dokuments:

```
<?xml version="1.0" encoding=" iso-8859-1">
<fo:root xmlns:fo=http://www.w3.org/1999/XSL/Format>
        <fo:layout-master-set>
        …
        </fo:layout-master-set>
        <fo:page-sequence>
        …
        </ fo:page-sequence>
        <fo:page-sequence>
        …
        </ fo:page-sequence>
<fo:root>
```

Seitendefinitionen mit <fo:layout-master-set>

Innerhalb von <fo:layout-master-set> werden die Maße und der Aufbau von Seiten mit <fo:simple-page-master> Elementen festgelegt. Es werden die Maße der Seiten, die Ränder der Ober- und Unterkanten geregelt. Weiterhin wird das FO-Dokument in Regionen unterteilt. Die Platzierung der Regionen auf einer Seite sieht man im Anhang auf Seite 37.

Die erste der fünf Regionen ist fo:region-before und entspricht der Kopfzeilen-Region. Die fo:region-after entspricht der Fußzeilen-Region. Desweiteren gibt es noch die fo:region-start und fo:region-end, die jeweils der linken und rechten Randspalte entsprechen. Und schließlich die größte Region ist die fo:region-body, innerhalb dessen die Inhalte platziert werden.

[15] vgl. Knobloch, M. & Kopp, M. (2001), S. 183ff

Definition der Seitenfolgen

Mit Hilfe der <fo:page-sequence-master> wird beschrieben, wie oft und in welcher Reihenfolge Seitentypen beim Einfüllen des Inhalts benutzt werden sollen. Innerhalb der <fo:page-sequence> gibt es weiter Elemente, um zum Beispiel zu regeln, welcher Inhalt auf welcher Fläche verteilt werden soll. Mit <fo:static-content> werden Inhalte platziert, die auf jeder Seite wiederholt werden müssen. Das sind beispielsweise Kopf- und Fußzeilen. Mit <fo:flow> wird der Fließtext bezeichnet, den es abhängig von seiner Länge auf eine oder mehrere Seiten zu verteilen gilt.

Das ganze zu programmieren ist sehr umfangreich und es selbst von Hand zu codieren nicht sehr sinnvoll.

7. Synchronized Multimedia Integration Language (SMIL)

SMIL ist eine auf XML basierende neue Auszeichnungssprache, die speziell für die Bedürfnisse von multimedialen Webelementen entwickelt wurde.

Durch SMIL wird es erleichtert Video, Text, Sound oder Grafiken miteinander zu verbinden und zu synchronisieren.

In HTML ist es nicht möglich, verschiedene Elemente aufeinander abgestimmt ablaufen zu lassen doch SMIL macht dies nun möglich.

Videos oder Sound wurden bisher als separate und isolierte Objekte betrachtet und behandelt. Auch für SMIL sind diese Objekte separat aufzubereiten, jedoch können sie mit der Synchronized Multimedia Integration Language aufeinander abgestimmt in einer einer „.smil“- Datei miteinander kombiniert werden.

Es lassen sich folglich verschiedene Videosequenzen, Bild-, Ton- und Textinhalte zeitlich aufeinander abgestimmt anzeigen.

Bisher waren solche Interaktionen zwar auch schon möglich, Sie benötigten aber nicht- standardisierte Plug-Ins oder Erweiterungen.

Ein großer Nachteil war zudem, dass ein einziges riesiges Datenobjekt geliefert wurde. Mit SMIL bleiben die einzelnen Elemente wie bei HTML üblich voneinander getrennt und können von unterschiedlichen URLs bezogen werden.

Zusätzlich gibt es die Möglichkeit unterschiedliche Bandbreiten des Benutzers zu berücksichtigen. Je nach verfügbarer Übertragungsgeschwindigkeit werden daran angepasste Objekte geliefert. Die Auswahl erledigt SMIL automatisch.
SMIL kommt mit sehr wenigen Befehlen aus und ist vergleichsweise einfach einzusetzen. Zudem ist die SMIL- Schnittstelle frei zugänglich und bleibt so nicht an einen einzigen Hersteller gebunden. Die Befehle sind an HTML angelehnt und aus diesem Grund leicht zugänglich.

7.1 Grundgerüst einer SMIL- Datei

```
<smil>
<head>
<meta name="copyright" content="Inhalt" />
<layout>
<!—Layout-Elemente-->
</layout>
</head>
<body>
<!—Media- und Synchronisation-Elemente -->
</body>
</smil>
```

Im *<body>* der SMIL-Datei werden die Inhalte eingebunden. Erlaubt sind RealMedia- und Bilddateien. Die Dateien werden mit den folgenden Tags eingefügt:

<audio/>
RealAudio Datei
<video/>
Realvideo Datei
<img/>
Bilddatei (GIF, JPEG, PNG)
<ref/>

alle Mediendateien, auch die erwähnten Dateien

<text/>

Textdateien

<textstream/>

RealText Dateien

<animation/>

Swf- Dateien

7.2 Fenster und Regionen

Fenster (root-layout)

Für alle visuellen Elemente, die in ein SMIL-Dokument eingefügt werden, ist es wichtig, dass diese innerhalb eines bestimmten Bereichs eingefügt werden.
Das so genannte root-layout enthält die gesamte Größe der SMIL Anwendung und stellt die Ausgabegröße der Präsentation dar.

Regionen bestimmen

Jedes einzelne Videoelement wird in einer eigenen Region ausgeführt.
Die Abmessungen und die relative Position dieser Region wird im Kopf des Dokuments angegeben.
Zusätzlich muss jedem Element eine eindeutige ***id*** zugewiesen werden.

7.3 Grafische Objekte einbinden

Nachdem die Regionen bestimmt sind, kann der eigentliche visuelle Inhalt der Datei angegeben werden. Dieser Inhalt befindet sich im body-Container des SMIL-Dokuments.

7.3.1 Darstellen von Objekten

7.3.1.1 Paralleles Abspielen der Objekte

Alle Elemente die durch das <par>- Tag umschlossen sind, werden parallel wiedergegeben.
Ein Beispiel befindet sich im Anhang auf der Seite 38.

7.3.1.2 Sequentielles Abspielen der Objekte

Elemente, die durch das Tag <seq> eingeschlossen sind, werden genau in dieser Reihenfolge, nacheinander wiedergegeben.
Ein Beispiel befindet sich im Anhang auf der Seite 37.

7.3.2 Dauer der Anzeige

Die Anzeigedauer kann durch den Befehl „dur=“Zeitangabe“ mit in dem <img src>- Tag eingebunden werden. Dieser Befehl ermöglicht es die einzelnen Elemente Sekundengenau zu steuern und perfekt abgestimmte Gesamtpräsentation aus einzelnen Quelldateien zu realisieren.
Ein Beispiel für die Dauer der Anzeige befindet sich im Anhang auf der Seite 37.

7.4 Beispiele

Ein Beispiel für eine sequentielle beziehungsweise eine parallele Darstellung von Mutimedia- Objekten befinden sich im Anhang auf Seite 38.
Der Inhalt der Datei smil2.smil zeigt, dass auch die Verschachtelung von den Tags <seq> und <par> möglich ist.

8. XML Linking Language (XLL)

Es existieren zwei parallele Linking-Entwürfe für XML um Verweise beschreiben zu können. Der erste Entwurf, Xlink, hat zwei Untergruppen. Die einfachen Links und die erweiterten Links.

8.1 Xlink

8.1.1 einfache Xlinks

Einfache Xlinks entsprechen ungefähr den Hyperlinkfähigkeiten von HTML. Er ist nur in eine Richtung möglich und beginnt dort an der Stelle im Dokument, wo er vorkommt.

Durch das Attribut Xlink:type=“simple“ wird dem XLink-Prozessor mitgeteilt, dass es sich um einen einfachen XLink handelt. Sobald nun der XLink-Prozessor einen einfachen Link gefunden hat, sucht er nach weiteren Attributen, die in Verbindung mit einfachen Links genutzt werden können.
Nachstehend werden nun die Attribute für einfache Links dargestellt und erläutert.

Xlink:href

Sein Wert ist der URI des anderen Endes des Verweises. Es kann sich auf ein ganzes Dokument als auch auf einen Punkt oder ein Element innerhalb jenes Dokumentes beziehen.
Dieses Attribut ist Pflicht, da sonst der Verweis keinen Sinn hat. Es muss folglich angegeben werden.

Xlink:role

Das Attribut „role“ beschreibt die Art des bjektes am anderen Ende des Verweises. Mittels eines Stylesheets können den unterschiedlichen Rollen verschiedene physische Auszeichnungen zuweisen.

Xlink:title

Dieses Attribut stellt einen Titel für die Ressource am anderen Ende des Verweises bereit. Dieses Attribut ist identisch mit dem title- Attribut im <a>-Element von HTML.

Xlink:show

Das Attribut „show“ schlägt der Applikation vor, was zu tun ist, wenn dem Verweis gefolgt wird.
Das Attribut „Show“ kann folgende Werte annehmen:

- „embed“: Der Inhalt, am anderen Ende des Verweises, wird geladen und an der Stelle, wo der Verweis steht, angezeigt.
- „replace“: Der Browser ersetzt die aktuelle Ansicht mit dem Ziel des Verweises, sobald darauf geklickt wird. Zu vergleichen ist dieses Attribut mit dem <a>-Element in HTML
- „new“: Der Browser soll einen neuen Kontext schaffen. Zum Beispiel wird ein neues Fernster geöffnet.

- „other“: Es liegt in der Hand der Applikation, wie der Verweis dargestellt wird.
- „none“: Durch diesen Wert wird kein Verhalten bestimmt.

Xlink:actuate

Dieses Verhaltensattribut legt fest, wann der Verweis aktiviert wird. Folgende Werte sind möglich:

- „onRequest“: Die Applikation wartet bis der Benutzer dem Verweis folgt.
- „onLoad“: Die Applikation folgt sofort dem Verweis, sobald sie auf ihn trifft. (mit dem <img>-Element in HTML zu vergleichen)
- „other“: Der Verweis wird auf eine Weise aktiviert, die nicht duch XLink vorgegeben ist. Andere Markups finden hier im Dokument anwendung
- „none“: Für den Fall der Aktivierung des Verweises stehen keine Informationen bereit. Er wird benutzt, wenn der Verweis kein gegenwärtiges sinnvolles Ziel besitzt.

8.1.2 erweiterte Xlinks

Die erweiterten Links ermöglichen Verweise zu mehreren Dokumenten.[16]

Durch das Attribut xlink:type=“extended“ wird der XLink-Applikation verdeutlicht, dass es sich um erweiterte Links handelt.

Erweiterte Verweise können zusätzlich einen von vier Typen besitzen. Diese werden folgendermaßen definiert: xlink:type=“Typ“

Anstelle von Typ können folgende Attribute treten:

- „resource“: Diese Angabe stellt die lokale Ressource für den Verweis zur Verfügung.
- „locator“: Stellt ein URI für das entfernte Dokument, was an dem Verweis beteiligt ist, zu Verfügung.
- „arc“: stellt eine Beschreibung der potentiellen Pfade unter den Dokumenten zur Verfügung, die an dem erweiterten Verweis beteiligt sind.
- „title“: bietet ein Label für den Verweis.

[16] zur Vertiefung siehe: Eckstein R. & Casabianca M. 2002, S. 99

8.2 Xpointer

Xpointer wurde dafür entwickelt, das Problem der Lokalisierung eines Elements oder eines Bereiches von Elementen in einem XML-Dokument zu lösen.[17]
Somit ermöglicht es XPointer Punkte, Bereiche und Positionen zu bestimmen und zu manipulieren.
Xpointer ist wesentlich flexibler als erweiterte Links mit Xlink. Es wird der Zugriff auf einzelne Objekte der Seite ermöglicht und es lassen sich Verweise auf exakte Bereiche eines XML-Dokuments beschränken.

9. Zusammenfassung/Ausblick

Zusammenfassend lässt sich sagen, dass die Vorzüge von XML unbestritten sind. Hierzu seien nochmals zwei Begriffe genannt, die XML und seine Vorteile insbesondere auch für die Gestaltung von Webseiten versinnbildlichen. Das ist zum einen der Begriff „Offenheit“ und zum anderen der Begriff „Standardisierung“[18]. Mit „Offenheit“ ist dabei gemeint, dass lauter Auszeichnungssprachen entstehen, die leicht durchschaubar sind und daher auch von vielen genutzt werden können. Dies birgt eigentlich die Gefahr, dass unterschiedliche Welten entstehen, die auseinanderdrifften. Da jedoch alle Auszeichnungssprachen auf Grundlage von XML definiert sind, ist sichergestellt, dass die nötigen Gemeinsamkeiten gegeben sind. Dies wird wiederum mit dem Begriff „Standardisierung“ zum Ausdruck gebracht. So lange wie eine Auszeichnungssprache also auf Grundlage von XML definiert ist, besteht keine Sorge über die Kompatibilität bzw. die Möglichkeiten für den Datenaustausch.
Der vielleicht einzige Nachteil von XML besteht darin, dass XML-Dateien fast immer größer sind als vergleichbare binäre Formate. Dieser Fakt wird jedoch relativiert, da zum einen Plattenplatz heutzutage nicht mehr so teuer ist und zum anderen es mittlerweile Programme wie zip gibt, die Dateien sehr gut und sehr schnell komprimieren können, so dass ebenso effektiv Bandbreite gespart werden kann.

[17] zur Vertiefung siehe: Eckstein R. & Casabianca M. 2002, S. 91
[18] vgl. Dünhölter, Kuno (1998)

XML kann überall da zum Einsatz kommen, wo Bedarf für Datenaustausch besteht. Die Zahl möglicher Anwendungen scheint letztlich unbegrenzt zu sein, da jeder seine eigene an seine Bedürfnisse angepasste Auszeichnungssprache schaffen kann. Daher wird es vermutlich auch nicht allzu lang dauern, bis XML genauso verbreitet ist wie HTML. Mittlerweile wird XML auch schon von allen großen Browsern unterstützt (Mozilla, Firefox, Netscape Navigator ab Version 6.0, Internet Explorer ab Version 5.0). Jedoch sei am Schluss nochmals hervorgehoben, dass XML kein Nachfolger von HTML ist und es auch nicht werden wird. Vielmehr bildet XML die Grundlage für eine neue Form des ursprünglichen HTML, welches auf SGML beruht, und zwar ist es das XHTML. Dies macht deutlich, dass HTML durch XML erweiterbar ist und XML daher eher als Ergänzung zu HTML gesehen werden kann. Abschließend lässt sich festhalten, dass XML zwar nicht immer die beste Lösung ist, es sich aber immer lohnt, XML in Erwägung zu ziehen oder wie es im Originalton des World Wide Web Consortium (W3C) heißt: „XML isn´t always the best solution, but it´s always worth considering.“[19]

[19] W3C (2003)

10. Quellen

Behme, Henning/ Mintert, Stefan (1998): *XML in der Praxis Professionelles Web-Publishing mit der Extensible Markup Language*. Addison Wesley Longman Verlag GmbH, Bonn

Eckstein, Robert/ Casabianca, Michel (2003): *XML kurz & gut*. O'Reilly Verlag GmbH & Co. KG, Köln

Harms, Florian/Koch, Daniel/Kürten, Oliver(2000): *Das große Buch (X)HTML & XML*. Data Becker GmbH & Co. KG, Düsseldorf

Knobloch, Manfred/ Kopp, Matthias (2001): *Web-Design mit XML*. dpunkt.verlag, Heidelberg

Kobert, Thomas (1999): *XML- Das bhv Taschenbuch*. bhv Verlags GmbH, Kaarst

Tolksdorf, Robert (2003): *HTML & XHTML- die Sprachen des Web*. dpunkt.verlag, Heidelberg

Dobnig, Mario (2000): *Einführung: XML verstehen.* Online im Internet:
URL: http://www.go4xml.com/xml/index.php [04.11.2004]
Dünhölter, Kuno (1998): *Das Web automatisieren mit XML*. Online im Internet:
URL: http://members.aol.com/xmldoku#2 [10.11.2004]
Keitz, Wolfgang v.(2000): *SMIL - Grundlagen und Anwendungsbeispiele*. Online im Internet: URL:http://v.hbi-stuttgart.de/~keitz/skripte/SMILStart.htm
Näf, Michael (2004): *Einführung in XML, DTDund XSL*. Online im Internet:
URL: http://www.internet-kompetenz.ch/xml/einführung [10.11.2004]
Schön, Dr. Eckhardt (2004): *XML für Medientechnologie*. Online im Internet:
URL:http://www.imt.tu-ilmenau.de/lehre/xml_mt/Folien_komplett.pdf
[21.11.2004]
SELFHTML e.V. (2001): *Darstellung von XML-Daten.* Online im Internet:
URL: http://de.selfhtml.org/xml/darstellung/index.html [26.11.2004]
W3C (2003): *XML in 10 points.* Online im Internet:

URL: http://www.w3.org/XML/1999/XML-in-10-points [22.11.2004]

Wikimedia Foundation Inc. (2004): *Extensible Markup Language.* Online im Internet:

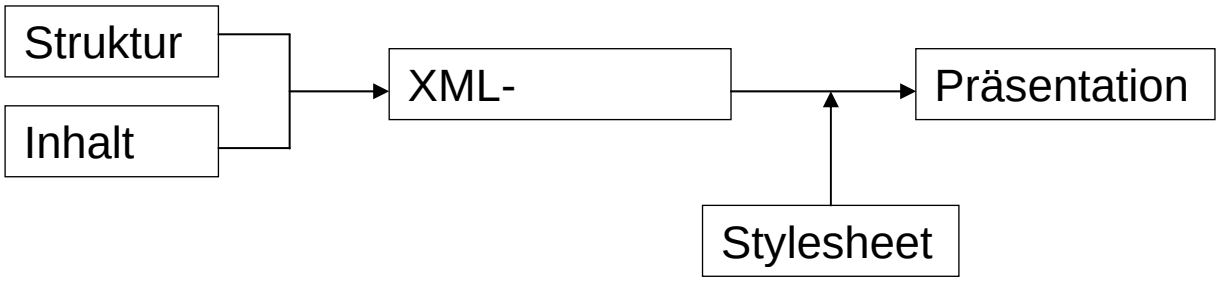
URL: http://de.wikipedia.org/wiki/XML [22.11.2004]

11. Anhang:

Ziele von XML:

1. einfache Nutzung für Internet
2. Unterstützung einer Vielzahl von Anwendungen
3. Kompatibilität zu SGML
4. einfaches Schreiben von Programmen, die XML-Dokumente bearbeiten
5. minimale Zahl optionaler Merkmale in XML (idealerweise Null)
6. XML-Dokumente für Menschen lesbar und angemessen verständlich
7. zügiges Abfassen des XML-Entwurfs
8. formaler und präziser Entwurf von XML
9. leichte Erstellung von XML-Dokumenten
10. Knappheit von XML-Markup ist von minimaler Bedeutung

Trennung von Struktur und Layout



(vgl. Schön, Dr. Eckhardt)

Inhalt der Datei adresse.xml:

```
<?xml:stylesheet type="text/css" href="adresse.css"?>
<dokument>

<titel>Adressliste</titel>
<autor>referatsgruppe XML und Webgestaltung</autor>


        <adresse>
                <Vorname>Sandra</Vorname>
                <nachname>Finger</nachname>
```

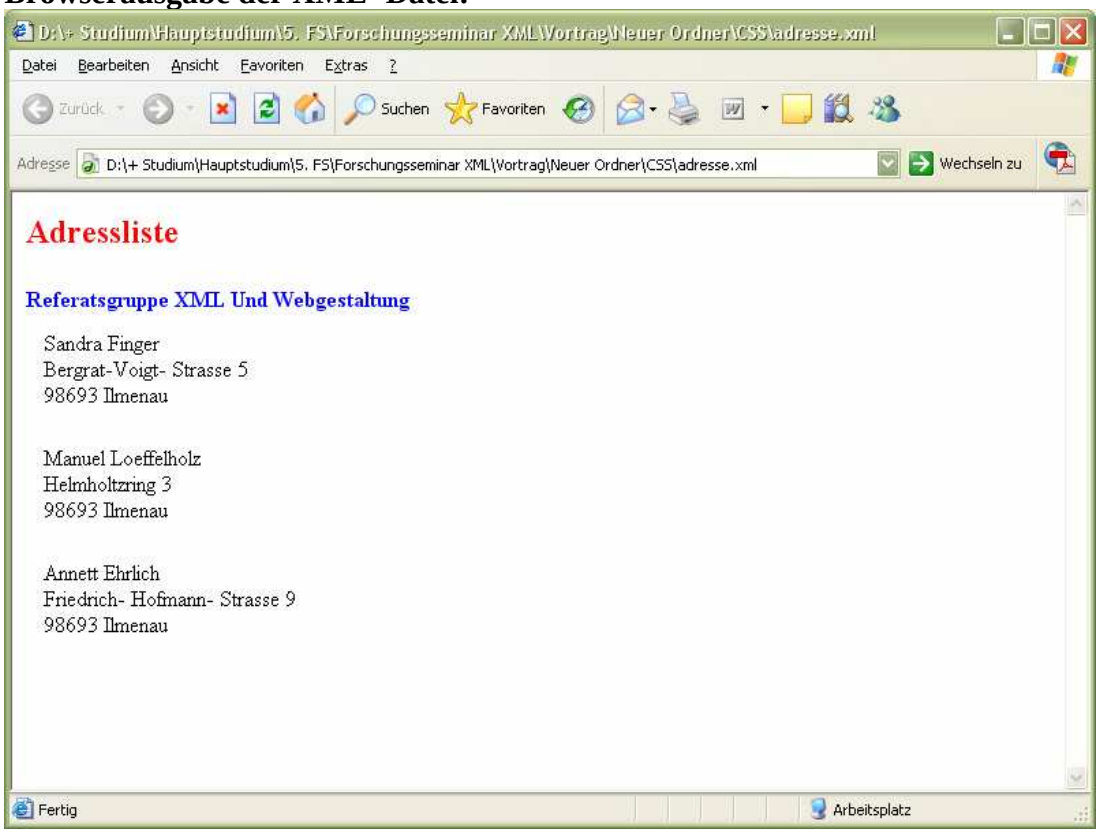
```
                <strasse>
                        <strassenname>Bergrat-Voigt- Strasse</strassenname>
                        <hausnummer>5</hausnummer>
                </strasse>
                <postleitzahl>98693</postleitzahl>
                <ort>Ilmenau</ort>
        </adresse>



        <adresse>
                <vorname>Manuel</vorname>
                <nachname>Loeffelholz</nachname>
                <strasse>
                        <strassenname>Helmholtzring</strassenname>
                        <hausnummer>3</hausnummer>
                </strasse>
                <postleitzahl>98693</postleitzahl>
                <ort>Ilmenau</ort>
        </adresse>



        <adresse>
                <vorname>Annett</vorname>
                <nachname>Ehrlich</nachname>
                <strasse>
                        <strassenname>Friedrich- Hofmann- Strasse</strassenname>
                        <hausnummer>9</hausnummer>
                </strasse>
                <postleitzahl>98693</postleitzahl>
                <ort>Ilmenau</ort>
        </adresse>

</dokument>
```

Inhalt der Datei adresse.css:

```
TITEL
    { display: block;
        font-size: 150%;
        font-weight: bolder;
        text-align: left;
        color: red;
        padding-top: 1pt;
        padding-bottom: 20pt
        }

AUTOR
    {   display: block;
        font-weight: bolder;
        text-transform: capitalize;
        text-align: left;
        color: blue;
        padding-bottom: 10pt
```

```
        }

adresse
    {   display: block;
        text-align: left;
        text-indent: 10pt;
        line-height: 1.2;
        padding-bottom: 20pt
        }

strasse {display: block;}
```

Browserausgabe der XML- Datei:



Inhalt der Datei Adresse.xsl:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
        <xsl:template match="/">
        <html>
                <TABLE border="4pt" border-color="yellow">
                        <TR STYLE="font-size:20pt; color:red">
                                <TD>Vorname</TD>
                                <TD>Nachname</TD>
                                <TD>Strasse</TD>
                                <TD>Hausnummer</TD>
                                <TD>Postleitzahl</TD>
                                <TD>Ort</TD>
                        </TR>
                        <xsl:apply-templates/> <!-- wendet alle Templates dieses Stylesheets
rekursiv auf das gegenwärtige Knotenelement an(Knotenelement ist in unserem Fall das
Wurzelelement "/") -->
                </TABLE>
```

```
	</html>
	</xsl:template>

<xsl:template  match="Dokument">
	<xsl:for-each select="Adresse">
		<TR STYLE="font-size:14pt; color:blue">
			<TD>
				<xsl:value-of select="Vorname"/>
			</TD>
			<TD>
				<xsl:value-of select="Nachname"/>
			</TD>
			<TD>
				<xsl:value-of select="Strasse"/>
			</TD>
			<TD>
				<xsl:value-of select="Hausnummer"/>
			</TD>
			<TD>
				<xsl:value-of select="Postleitzahl"/>
			</TD>
			<TD>
				<xsl:value-of select="Ort"/>
			</TD>
		</TR>
	</xsl:for-each>

</xsl:template>


</xsl:stylesheet>
```

Inhalt der Datei choose.xsl:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
	<h3>Die Postleitzahlen lauten:</h3>
	<xsl:for-each select="Dokument/Adresse/Postleitzahl">
	<div>
		<xsl:value-of select="." />
		<xsl:variable name="wert" select="." />
			<xsl:choose>
			<xsl:when test="$wert < 10000">
			<xsl:text>(unvollständige Postleitzahl)</xsl:text>
			</xsl:when>
			<xsl:otherwise>
			<xsl:text />
			</xsl:otherwise>
```

```
                    </xsl:choose>
            </div>
            </xsl:for-each>
    </body>
    </html>
    </xsl:template>
 </xsl:stylesheet>
```

Inhalt der Datei comment.xsl:

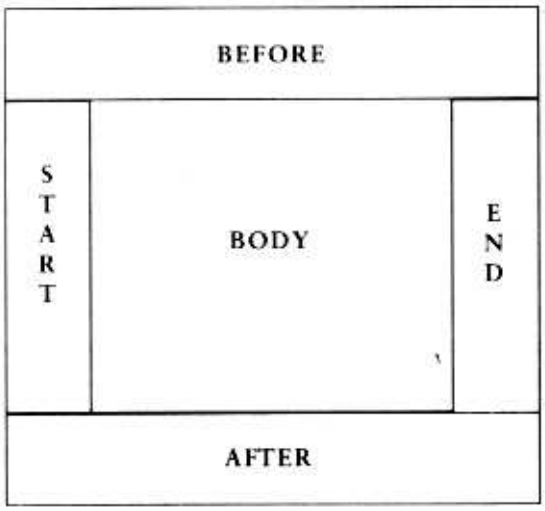
```
    <?xml version="1.0" encoding="iso-8859-1" ?>
    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
            <xsl:template match="/">
            <html>
            <head>
                    <script language="JavaScript" type="text/javascript">
                    <xsl:comment>
                    alert ("Und schon erscheint ein Kommentar!")
                    </xsl:comment>
                    </script>
            </head>
            <body>
                    Kommentar
            </body>
            </html>
            </xsl:template>
    </xsl:stylesheet>
```

Inhalt der Datei sort.xsl:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<table border="1">
<xsl:for-each select="Dokument/Adresse">
<xsl:sort select="Hausnummer" order="descending" datatype="number" />
<tr> <td>
                    <xsl:value-of select="Vorname" />
```

```
            </td> <td>
<xsl:value-of select="Hausnummer" />
            </td>     </tr>
            </xsl:for-each>
            </table>
    </body>
    </html>
    </xsl:template>
  </xsl:stylesheet>
```

Regionen eine XSL:FO Dokuments:



Inhalt der Datei smil1.smil:

```
<smil>
<head>
        <layout>
                <root-layout width="740" height="580"
                background-color="black" />

                <region id="title" left="320" top="255"
                width="90" height="20"/>
                <region id="region_1" left="75" top="50"
                width="350" height="303" />
                <region id="region_2" left="350" top="50"
                width="200" height="148" />
                <region id="region_3" left="400" top="220"
                width="300" height="222" />
                <region id="video_region" left="180" top="140"
                width="440" height="280"/>

        </layout>
</head>

<body>
        <seq>
```

```
                <text src="title.txt" region="title" dur="5s"/>
                <img src="bild1.jpg"  region="region_1" dur="3s"/>
                <img src="bild2.jpg"  region="region_2" dur="3s"/>
                <img src="bild3.gif"  region="region_3" dur="3s"/>
                <video src="Papa.mpg" region="video_region"/>
        </seq>
</body>

</smil>
```

Inhalt der Datei smil2.smil:

```
<smil>
        <head>
                <layout>
                <root-layout width="740" height="580"
                background-color="white" />

                <region id="title" left="550" top="255"
                width="110" height="20"/>
                <region id="region_1" left="10" top="10"
                width="350" height="303" />
                <region id="region_2" left="10" top="350"
                width="200" height="148" />
                <region id="region_3" left="388" top="280"
                width="300" height="222" />
                <region id="video_region" left="388" top="10"
                width="300" height="240" />
                </layout>
        </head>

        <body>
           <seq>
                <par>
                   <img src="bild1.jpg" region="region_1" dur="3s"/>
                   <img src="bild2.jpg" region="region_2" dur="5s"/>
                   <img src="bild3.gif" region="region_3" dur="6s"/>
                   <video src="Video2.mpeg" region="video_region"/>
                   <text src="title.txt" region="title" dur="22s"/>
                </par>
             <audio src="sound1.wav"/>
          </seq>


        </body>
</smil>
```