# XML als Quelle von Web-Seiten

## Eine Handreichung zum XML/XSL – Workshop am 8.12.1999

Autor: M. Knobloch
Kompetenzzentrum für Multimedia und Telematik
Konrad Adenauer Str. 40
72072 Tübingen

## Vorbemerkung:

Die Dokumente des WWW-Konsortiums, auf die im Skript mehrfach Bezug genommen wird, finden sich unter http://www.w3.org. Eine detailliertere Angabe zu einzelnen Dokumenten finden Sie im Anhang.

Die verwendeten Beispiele wurden unter Windows NT, Java 1.2, teilweise auch unter Linux mit Java 1.1.7 getestet. Als XML-Parser und XSL-Prozessoren wurden - zumindest für Forschung und Lehre - frei verfügbare Java Programme verwendet.

Die vornehmlich von Firma IBM stammenden Werkzeuge (XML4J und LotusXSL) sind mittlerweile Bestandteil des XML.APACHE-Projekts (http://xml.apache.org) geworden und sind dort unter den Namen XERCES und XALAN verfügbar. Es wird empfohlen, bei eigenen Experimenten diese Werkzeuge zu verwenden, da sie die Aussicht auf große Verbreitung haben.
Die verwendeten Beispiele sollten auch mit diesen Werkzeugen lauffähig sein. Allerdings müssen bei Verwendung dieser Werkzeuge die CLASSPATH Einträge und die Paketnamen der aufgerufenen Klassen der Apache-Namensgebung angepasst werden.

Im Text werden die englische Wörter *template* und *tag* kursiv und klein geschrieben - auch am Satzanfang. *tag* (Text-Auszeichnungselement) sollte nicht mit dem 24-Stunden-Zeitintervall verwechselt werden.

# INHALTSVERZEICHNIS

# 1 Einführung

Nachdem XML kurze Zeit für Schlagzeilen in jeder PC-Zeitschrift sorgte, scheint es derzeit eher still geworden zu sein. XML war keine Eintagsfliege. Die Softwareindustrie ist vielmehr im Begriff viele Anwendungen auf XML umzurüsten. Meistens ist das für Endanwender nicht leicht ersichtlich. Aus XML generiertes HTML steht dabei nicht so sehr im Zentrum des Interesses, wie XML als Austausch-, Definitions- oder Metadatenformat. An vielen Stellen wird einfach am praktischen Einsatz gearbeitet.
Dennoch, Daten wollen nicht nur transportiert und decodiert, sondern auch von Menschen konsumiert werden. Wie werden XML Strukturen lesbar? XML selbst hat keinerlei Möglichkeiten Steuersequenzen für Drucker oder Grafiktreiber mit sich zu führen. Mit XML werden Dokumente gegliedert, für die Darstellung ist XSL - extensible style language - zuständig.

# 2 XML

Markup

Zentrales Anliegen für den Einsatz von XML ist es, Inhalte maschinell zugänglich, auffindbar und manipulierbar zu machen. Um das zu erreichen, wird mit XML eine Möglichkeit gegeben, Inhalte über kennzeichnende Markierungen in funktionale Blöcke zu untergliedern. Diese Auszeichnungen werden *tags* genannt. *tags* werden durch spitze Klammern gekennzeichnet, z.B.: <kapitel>. Wird von der Gesamtheit der *tags* gesprochen, spricht man von *Markup*.
Die Definition von XML des World Wide Web Konsortiums (dt. Fassung: http://www.mintert.com/xml/trans/REC-xml-19980210-de.html) beschreibt eine Menge von Regeln, die festlegen wie *Markup* zur Dokumentengliederung eingesetzt werden kann. Auch hier werden lediglich Beschreibungsstrukturen festgelegt. Die Menge dieser Regeln wird Grammatik genannt. In diesem Sinn ist XML eine Beschreibungssprache.

tags

Die Menge der möglichen Auszeichnungen (*tags*), die zur Dokumentenstrukturierung in XML benutzt werden dürfen, ist nicht festgelegt. Neue Auszeichnungen und deren Funktion im jeweiligen Dokument können von Anwendungs- und DokumententwicklerInnen je nach Bedarf erfunden werden. Deshalb wird XML als erweiterbar bezeichnet und so erklärt sich der Name e**X**tensible **M**arkup **L**anguage - Erweiterbare Auszeichnungssprache. Genau genommen ist XML eine Metasprache, die es erlaubt, Markup-Sprachen zu definieren. Beispiele sind MathML, WAP, ...

neu an XML

Prinzipiell neu sind die Ideen, die in der Definition von XML stecken nicht. Als Vorläufer und Obermenge ist SGML bekannt. Neu ist die breite Unterstützung von XML als quasi Standard durch eine sehr große Zahl wichtiger Softwarehersteller.

## 2.1 Wohlgeformt und gültig

Wie sieht XML aus? Ein bißchen wie HTML und doch anders. Der wichtigste Unterschied: In XML sollten keine Anweisungen enthalten sein, die beschreiben, wie etwas am Bildschirm oder auf Papier dargestellt werden soll. Ein XML-Datenstrom sollte lediglich *tags* enthalten, die für die Strukturierung des Dokuments bedeutsam sind.

Beispiel:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE postkarte SYSTEM "postkarte.dtd">
  <postkarte datum="1.9.1999">
    <anschrift>
      <adressat>Manfred Knobloch</adressat>
      <strasse>Mathildenstr. 6</strasse>
      <plz>D-72072</plz>
      <ort>Tübingen</ort>
      <land>Germany</land>
    </anschrift>
    <text>
      <gruss>Liebe Freunde, </gruss>
      <textkorpus>uns geht es gut, das Wetter
        und das Essen sind prima,
        nur das Wasser ist zu kalt.
        Egal, den Kindern gefällt der viele
        Sand und wir haben unsere Ruhe.
      </textkorpus>
    </text>
    <mfg>Viele liebe Grüsse von Euren</mfg>
  </postkarte>
```

tags

Was ist mit dieser einfachen Postkarte geschehen? Die Textteile der Postkarte wurden mit Auszeichnungen versehen. Auszeichnungen, die eine Beschreibung über den bezeichneten Dokumententeil enthalten. Die in diesem Beispiel benutzten Regeln sind für jeden menschlichen Leser sofort verständlich:
Auszeichnungen (*tags*) kommen immer paarweise vor: einmal öffnend <postkarte>, um den Beginn einer Sequenz anzuzeigen und einmal schließend </postkarte>,

um das Ende eines Dokumententeils zu kennzeichnen.

leere Elemente

Leere Elemente sind auch möglich. Sie kommen im obigen Beispiel noch nicht vor, sollen aber an dieser Stelle genannt werden, da sie nicht unbedingt mit zwei *tags* notiert werden müssen. Für leere Elemente ist die Kurzschreibweise <postkarte/> erlaubt.

wohlgeformt

Weist ein Dokument zu jedem öffnenden auch ein schließendes *tag* auf und sind diese *tags* lediglich hierarchisch ineinander geschachtelt, spricht man von einem **wohlgeformten** XML-Dokument. Eine überlappende Schachtelung, wie folgendes Konstrukt, würde die Wohlgeformtheit zerstören:

```
<gruss>Liebe Freunde,
<textkorpus>uns geht es gut
</gruss>
</textkorpus>
```

Hier wird <gruss> geschlossen, bevor das eingebettete *tag* <textkorpus> beendet ist. Wohlgeformtheit ist erreicht, wenn folgende Bedingungen erfüllt sind:

| Zusammenfassung: Regeln für Wohlgeformtheit |
|---|
| • Dokumente bestehen aus Markup (*tags*) und Inhalt.<br>• Jedes Element hat ein Start- und ein Ende- *tag* oder besteht aus einem leeren Element ohne Inhalt.<br>• Im Start-*tag* eines Elements sind einmal vorkommende Informationen in Form von Attributen erlaubt.<br>• Elemente sind hierarchisch geschachtelt.<br>• Jedes XML-Dokument hat genau ein einschließendes Root-Element. |

DTD

Darüber hinaus ist es möglich, dem XML-Dokument eine Art Grammatik, ein Strukturschema, zuzuordnen. Die Zuordnung wird über die Zeile in der XML-Datei

```
<!DOCTYPE postkarte SYSTEM "postkarte.dtd">
```

erreicht. Diese Liste von Regeln, die beschreibt welche Elemente in diesem Dokument zulässig sind, wird Document Type Definition - kurz DTD - genannt. Eine DTD für unsere Postkarte könnte so aussehen:

```
<!ELEMENT postkarte (anschrift, text, mfg)>
<!ATTLIST postkarte datum CDATA #IMPLIED>
<!ELEMENT anschrift (adressat, strasse, plz, ort, land)>
<!ELEMENT adressat (#PCDATA)*>
<!ELEMENT strasse (#PCDATA)*>
<!ELEMENT plz (#PCDATA)*>
<!ELEMENT ort (#PCDATA)*>
<!ELEMENT land (#PCDATA)*>
<!ELEMENT text (gruss, textkorpus*)>
<!ELEMENT gruss (#PCDATA)*>
<!ELEMENT textkorpus (#PCDATA)*>
<!ELEMENT mfg (#PCDATA)*>
```

**gültig**

Entspricht das XML-Dokument den Regeln der zugeordneten DTD, spricht man von einem gültigen (valid) XML-Dokument.
Es fällt auf, daß DTDs nicht in XML-konformer Notation dargestellt werden und, daß es keine Typisierung der beschriebenen Daten gibt, d.h. alle Informationen als Zeichenketten behandelt werden. Dies ist einer der Hauptkritikpunkte an DTDs. Als Ersatz für DTDs werden derzeit sogenannte Schemata diskutiert, die diesem Mangel abhelfen sollen. Schemata werden in XML-Notation dargestellt.

**gültig oder wohlgeformt**

Ein XML-Dokument kann maschinell verarbeitet werden, auch wenn es keine DTD besitzt. Die Wohlgeformtheit genügt gängigen XML-Werkzeugen.
Zur Prüfung von Eingaben in ein XML-Dokument werden DTDs jedoch gebraucht. XML-Editoren bieten in der Regel an, ein Dokument beim Editieren automatisch oder bei bestimmten Benutzeraktionen zu validieren. Werkzeuge wie Xeena blenden nur Eingabefelder ein, die in der zugeordneten DTD an dieser Stelle zugelassen sind.

Im Beispiel werden diverse Schlüsselworte zur Beschreibung der Elemente bzw. der Attribute verwendet. Leider sind diese Definitionen in keiner Weise selbst erklärend und unterscheiden sich auch in der Notation hinsichtlich ihrer Gültigkeit. Für XML-Elemente und ihre Attribute gelten folgende Regeln:

| Zusammenfassung: Regeln für Element Inhalte | |
|---|---|
| • EMPTY | kein Inhalt |
| • ANY | beliebiger Inhalt |
| • \| | Trennzeichen für Auswahlliste |
| • , | Trennzeichen für Liste mit fester Reihenfolge |
| • ? | kein oder ein Vorkommen |
| • * | kein oder beliebig viele Vorkommen |
| • | genau ein Vorkommen (Leerzeichen) |
| • + | ein oder mehrere Vorkommen |
| • () | Gruppierung |
| • (#PCDATA) | parsed character data: beliebige Zeichen |
| • (#PCDATA \| ..) | mixed content: Zeichen oder weitere Kindelemente |

| Zusammenfassung: Regeln für Attribute | |
|---|---|
| • CDATA | Zeichen |
| • ID | Dokumentweit eindeutiger Wert |
| • IDREF(s) | Bezug, Referenz auf eine ID des Dokuments |
| • (Aufzählung) | Liste erlaubter Werte |
| • #REQUIRED | Wert muß angegeben werden |
| • #IMPLIED | Wert kann (muß nicht) angegeben werden |
| • "text" | Defaultwert |
| • #FIXED | Wert ist nicht veränderbar (konstant) |

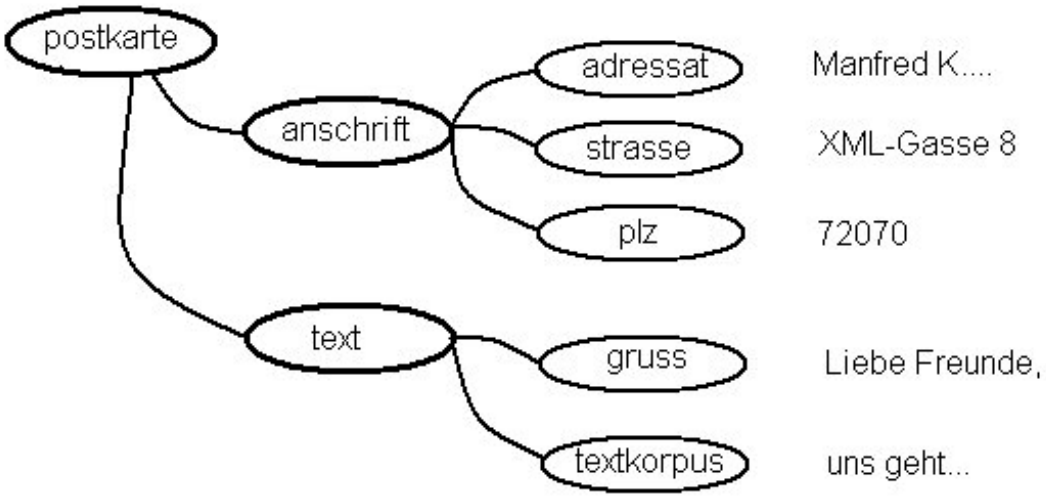
Unsere Beispielpostkarte hat das Attribut Datum. In der DTD wurde es als IMPLIED deklariert, folglich muß es nicht zwingend angegeben werden.

## 2.2 Die Rolle des Parsers

Wozu all der Aufwand? Für Menschen ist es leichter das Ganze ohne *tags* zu lesen. Und noch unnatürlicher scheint die Definition einer DTD.

XML wurde jedoch auch im Hinblick auf maschinelle Verarbeitung von Dokumenten erfunden. Schon die Festlegung der Wohlgeformtheit genügt, um einen Parser[1] einzusetzen, der ein solches Dokument strukturiert durcharbeiten kann.

[1] Ein Parser ist ein Programm, das grammatische Elemente in einem Dokument findet und bestimmt.




Ein Parser kann angewiesen werden auf der Ebene der direkten Nachkommen des Dokuments (document root <postkarte>) den Knoten <anschrift> auszuwählen, von diesem alle Kindelemente zu durchlaufen und deren textuellen Inhalt zu verarbeiten.
Oder man könnte nur die Grüsse extrahieren lassen.

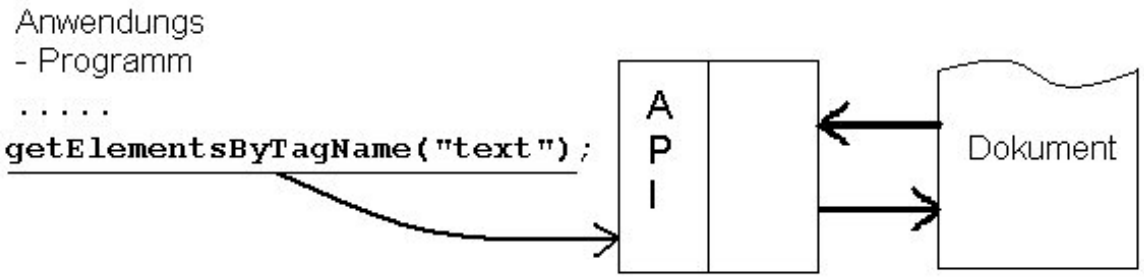
Diese Möglichkeit ist natürlich dann entscheidend, wenn nicht nur ein einziges Dokument der Art <postkarte>, sondern viele, vielleicht gemischt mit anderen Dokumenttypen (<brief>, <bericht>....) im Datenstrom vorhanden sind.

## Einsatzgebiete

Das Haupteinsatzgebiet von XML liegt jedoch nicht im Bereich der klassischen Textverarbeitung. XML-Dokumente entstehen derzeit vor allem in Umgebungen, in denen die Texte selbst von Programmen erzeugt werden oder in Bereichen, die mit strukturierten Daten arbeiten. Das heißt: Rechnungen und Bestellungen, Datenaustauschprotokolle zwischen Programmteilen, Teilebeschreibungen in der Industrie, Katalogdaten, die in verschiedene Ausgabeformate transformiert werden sollen. In solchen Zusammenhängen ist es oft notwendig, daß ein Parser überprüft, ob ein Dokument zu seiner DTD konform (valid) ist. Ein Parser, der die Gültigkeit eines Dokuments prüfen kann, heißt deshalb validierender Parser.

## 2.3 Arbeit mit einem Parser

Ein XML-Parser ist ein Programm, dem ein XML-Datenstrom übergeben wird. Der Parser kann beim Durchlesen des Dokuments eine Struktur aufbauen, die das Dokument im Arbeitsspeicher des Rechners in definierter Weise repräsentiert. Diese Definitionen sind von Anwendungsprogrammen als Methodenaufrufe nutzbar.




API

Diese Menge der Methodenaufrufe, die dem Anwendungsprogramm zur Verfügung gestellt wird, wird API - application programming interface - genannt. Ein Parser bietet also Zugriff auf ein Dokument in Form von APIs.

| Hinweis: |
| --- |
| Obgleich XML sprach- und plattformunabhängig konzipiert ist, sind die allermeisten verfügbaren Parser in Java realisiert. Weitere häufige Implementierungssprachen sind C/C++, Phyton und Perl. Es ist deshalb nicht ganz beliebig, in welcher Programmiersprache Anwendungsprogramme, die XML-Daten verarbeiten sollen, geschrieben sind.<br>Der Vorteil von fertigen Parsern liegt dennoch auf der Hand: Die gesamte Logik des Einlesens und Dekodierens des Dokuments ist nicht mehr Problem des Anwendungsprogramms. Das Programm wählt, ähnlich wie bei einer Datenbankanfrage nur die Teile des Dokuments aus, die es benutzen möchte. |

DOM & SAX

Viele aktuelle Parser bieten zwei APIs zum Zugriff auf XML-Dokumente an: DOM und SAX.
Auf die Unterschiede der beiden APIs wird im Folgenden kurz eingegangen.

### *2.3.1 DOM (Document Object Model)*

DOM steht für "Document Object Model". DOM ist die Beschreibung einer Menge von neutralen Schnittstellen zur Navigation in und zur Manipulation von Dokument-Objekten. DOM wurde vom WWW-Konsortium definiert und liegt als Standard in der Version 1 vor. Die erweiterte Version 2 ist in Bearbeitung.

Darstellung des Dokuments als Baumstruktur

Ein DOM-fähiger Parser erzeugt aus einer XML-Datenquelle ein DOM-Objekt, also ein Gebilde, das den Spezifikationen des WWW Konsortiums entspricht. DOM ist die Abstraktion vom realen Dokument - vergleichbar einer spezifischen Sichtweise. Ein DOM-Dokument ist eine Baumstruktur aus sogenannten Knoten (nodes). Ein DOM Parser stellt konsequenterweise ein Node-Interface zur Verfügung, das es einem Anwendungsprogramm erlaubt abzufragen, ob ein Knoten weitere Kindelemente enthält, oder ob es sich um einen Text-Knoten handelt und dergleichen mehr. Ein weiteres Interface ist das NodeList Interface, mit dem beispielsweise Iterationsmuster über eine Liste von Knoten durchgeführt werden können.

### *2.3.2 SAX*

Ein Dokument ist eine Folge von Ereignissen

Während DOM Operationen auf einem statischen Dokumentbaum beschreibt, verfolgt das Simple API for XML - SAX - einen anderen Weg. Beim Durchgang durch den XML-Datenstrom, wird jeder aktuell besuchte Knoten als Ereignis (event) betrachtet.

Rückruf aus dem Dokument

Das Anwendungsprogramm, das mit SAX auf ein Dokument zugreift, kann für jeden Knotentyp eine Funktion bzw. eine eigene Methode (listener) definieren, die ausgeführt wird, wenn das Ereignis „Knoten xy gefunden“ eintritt. Der Parser „ruft“ das Anwendungsprogramm, das ihn aufgerufen hat, quasi zurück; dieses meldet sich mit seiner (callback) Funktion. SAX stellt XML-Daten für ein Anwendungsprogramm nicht als Baum, sondern als Folge von Ereignissen (events) dar. Das Anwendungsprogramm muß Informationen aus dem Dokument selbst aufbewahren, die über das unmittelbare Auftreten des Ereignisses hinaus von Belang sind. SAX-parsing benötigt deshalb meist weniger Speicherplatz und glänzt durch schnelle Ausführungszeiten. Dafür ist eben der Aufwand im Anwendungsprogramm grösser.

DOM oder SAX?

Das API der Wahl ist DOM eher dann, wenn es sich bei

den Quelldaten um Dokumente im herkömmlichen Sinne handelt. Sollen maschinengenerierte und maschinenlesbare Daten wie Abfrageergebnisse, Listen etc. verarbeitet werden, ist SAX erwägenswert.

## 2.4 Warum XSL?

**Jenseits der Programmierung**

Die Bearbeitung von Dokumenten durch den Parser erfordert fast immer eine Programmierumgebung. Soll das API des Parsers ausgeschöpft werden, muß ein Anwendungsprogramm geschrieben werden.

**Stylesheets**

Sicher gibt es in der Praxis genügend Anwendungsfälle, in denen dies unabdingbar ist. XML wurde jedoch von Anfang an so konzipiert, daß die Beschreibung der graphischen Repräsentation der Daten nicht im Dokument, sondern in einer separaten Datei zu finden ist. In Anlehnung an vorhandene Techniken, wurde diese Datei StyleSheet genannt. Sie enthält Formatierungsanweisungen in XML-Notation.

**XSL-Prozessor**

Für die Verarbeitung der Formatierung von XML-Daten wurden deshalb Standardprogramme geschaffen, XSL-Prozessoren, die ein XSL-Stylesheet gegen XML-Quelldaten abarbeiten.

**XML->XSL**

Vom WWW-Konsortium wurde definiert, wie die Verknüpfung zwischen dem XML-Dokument und dem XSL-Stylesheet beschrieben wird (*Associating stylesheets with XML documents.* W3C Recommendation): Im XML-Dokument wird eine Processing-Instruction (pi) eingefügt, die eine Referenz auf das Stylesheet enthält:

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

Diese Methode stellt sicher, daß zu einem derart gekennzeichneten XML-Dokument Verarbeitungsregeln maschinell gefunden und abgearbeitet werden können.

# 3 XSL

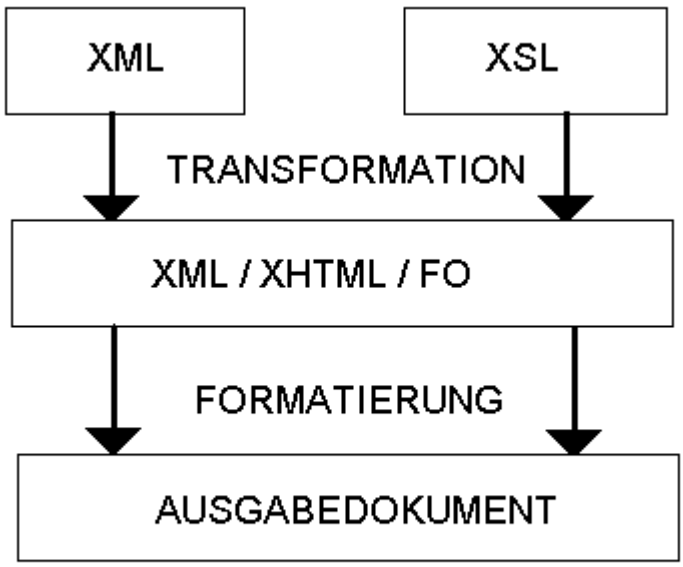
XSL ist eine Sprache zur Beschreibung von Gestaltungsanweisungen, die in Dateien als sog. Stylesheets abgelegt werden. Entwickler und Gestalter verwenden XSL, um die Repräsentationen von XML-strukturierten Daten zu definieren. Dies betrifft die Gestaltung einer Ausgabe der Daten auf einem Medium wie etwa einem Fenster in einem Web-Browser, eine Anzahl von Seiten in einem Buch, ein Report und dergleichen mehr.

XSL-Prozessor

Ein Stylesheet-Prozessor ist ein Programm, das XML-Daten und ein Stylesheet als Eingabe empfängt und daraus eine Repräsentation erzeugt, die den Anweisungen des Stylesheets genügt.

Transformation und Formatierung

Die Verarbeitung, die ein Stylesheet-Prozessor durchführt gliedert sich in zwei Teilprozesse auf. Zunächst wird ein Ergebnisbaum aus dem XML-Quellbaum erzeugt. Im zweiten Durchgang wird dieser Ergebnisbaum interpretiert, um eine formatierte Ausgabe - auf Bildschirm, Papier, Sprache oder andere Medien - zu erzeugen.
Der erste Subprozess wird Dokumentbaum-*Transformation* (tree transformation oder tree construction), der zweite Prozess *Formatierung* genannt.




Diese Aufspaltung in zwei Prozesse ermöglicht eine große

Flexibilität bei der Erzeugung der Repräsentation der Quelldaten.

## 3.1 Die Transformation (XSLT und XPath)

Während der Transformation kann die Struktur des Transformations-Baums vollständig abweichend von der des Eingabedokuments erzeugt werden. Die Transformation kann als Filter fungieren, die Dokumentanordnung reorganisieren, sortieren oder auch Elemente hinzufügen. Beispielsweise kann ein Inhaltsverzeichnis aus einem filternden Durchlauf durch das Quelldokument erzeugt werden.

Die Beschreibung der Transformation hat das WWW-Konsortium in den Dokumenten zu

- XSL Transformations (XSLT)
- XML Path Language (XPath)

niedergelegt. Die Spezifikation zu XPath wurde aus dem Transformationsdokument ausgegliedert. Dieses Dokument befaßt sich mit der Navigation in und der Auswahl von Elementen aus dem XML-Quelldokument. XPath verwendet eine Syntax, die der von Pfadangaben in Dateisystemen ähnelt, jedoch selbst nicht XML-konform ist. Auf die Mechanismen der Transformation wird nachfolgend noch genauer eingegangen.

## 3.2 XSL und die Formatting Objects

formatting objects

Der Formatierungsprozess selbst wird ermöglicht, indem der Ergebnisbaum der Transformation mit Elementen einer Formatierungssemantik versehen wird. XSL nennt diese Elemente Formatierungsobjekte (formatting objects). Die Knoten des Ergebnisbaums sind Instanzen von formatting objects (FO). FOs beschreiben typographische Abstraktionen wie Seite, Abschnitt, Positionsregeln, etc. Genauere Beschreibungen dieser Abstraktionen werden durch eine Menge von zuordenbaren Eigenschaften (properties) ermöglicht. Dies betrifft Einzüge, Wort und Buchstabenzwischenräume und dergleichen mehr.

Der Formatierungsprozess erzeugt aus diesem FO-Datenstrom ein Endprodukt wie z.B. PDF-Dateien. Formatierer für spezielle Ausgabemedien sind derzeit noch nicht für den praktischen Einsatz verfügbar. Im experimentellen Stadium verfügbar ist FOP - "formatting objects for pdf".

Die Beschreibung der Formatierung und der formatting objects hat das WWW-Konsortium im Dokument

- Extensible Stylesheet Language (XSL)

niedergelegt. Es liegt derzeit auf dem Stand von April 1999 als working draft vor und ist bereits jetzt mit über zweihundert Seiten das umfangreichste Dokument der *XSL*-Dokumentenfamilie.

### *3.2.1 Beispiel zu XSL:FO*

<xsl:fo ...> Das Vokabular einer Formatierung besteht aus Elementknoten des Ergebnisbaums, die eine definierte Menge von xsl:fo-Elementen enthalten.

<fo:root> Jedes Formatierungsdokument enthält ein fo:root Element, in das ein oder mehrere layout-master-set und page-sequence Elemente einbezogen sind.

layout-master-set Im layout-master-set werden allgemeine Angaben zum Seitenaufbau abgelegt. Diese Angaben haben Definitionscharakter. Sie können benannt und im Verlauf der Formatierung aufgerufen und angewendet werden.

page-sequence Im page-sequence Abschnitt werden über die Definition der Seitenabfolgen die Seitendefinitionen adressiert und im sogenannten fo:flow Block die eigentlichen Arbeitsdaten positioniert und formatiert.

Nachfolgender Auszug aus einem Stylesheet erzeugt eine FO-Datei. Es wird als Beispiel von J.Tauber mit dem FOP Prozessor mitgeliefert.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
             xmlns:fo="http://www.w3.org/XSL/Format/1.0">
  <xsl:template match="novel">
     <fo:root>

       <fo:layout-master-set>
          <fo:simple-page-master
               page-master-name="right"
               margin-top="75pt"
               margin-bottom="25pt"
               margin-left="100pt"
               margin-right="50pt">
            <fo:region-body margin-bottom="50pt"></fo:region-body>
            <fo:region-after extent="25pt"></fo:region-after>
          </fo:simple-page-master>
          <fo:simple-page-master
               page-master-name="left"
               .........
               margin-right="100pt">
            ........
          </fo:simple-page-master>
       </fo:layout-master-set>

       <fo:page-sequence>
          <fo:sequence-specification>
            <fo:sequence-specifier-alternating page-master-
first="right" page-master-odd="right" page-master-
even="left"></fo:sequence-specifier-alternating>
          </fo:sequence-specification>

          <fo:static-content flow-name="xsl-after">
            <fo:block text-align-last="centered" font-size="10pt">
              <fo:page-number></fo:page-number>
            </fo:block>
          </fo:static-content>

          <fo:flow>
            <xsl:apply-templates></xsl:apply-templates>
          </fo:flow>

       </fo:page-sequence>
     </fo:root>
  </xsl:template>

  <xsl:template match="front/title">
     <fo:block font-size="36pt" text-align-last="centered" ...>
       <xsl:apply-templates/>
     </fo:block>
  </xsl:template>
  <xsl:template match="chapter">
     <xsl:apply-templates/>
  </xsl:template>
....
</xsl:stylesheet>
```

(Auszug aus dem sample fo-stylesheet zur Definition Seitenaufbau und Folge)

# 4 XSLT Transformation von XML zu XML

XSLT definiert Regeln, wie ein XML-Dokument in ein anderes überführt werden kann. Das neu entstehende Dokument kann seinerseits Markup und DTD des Quelldokuments enthalten oder auch eine Menge vollständig unterschiedlicher *tags* verwenden. Eventuell kann das Zieldokument *tags* enthalten, die XSL-formatting objects enthalten.

## 4.1 Der prinzipielle Ablauf einer XSLT Verarbeitung

| | |
|---|---|
| | Im Verlauf einer Transformation liest der XSL-Prozessor die XML-Datei und das XSL-Stylesheet. Abhängig von den darin enthaltenen Anweisungen generiert der XSL-Prozessor ein vollständiges neues XML-Dokument oder ein Fragment als Ausgabe. Fragment bedeutet hierbei, daß das neue Dokument wohlgeformt wäre, wenn eine umschließende Klammer von *tags* hinzugefügt würde. Dies macht Sinn in Anwendungsfällen, in denen das Fragment als *parsed entity* in ein anderes Dokument integriert werden soll. |
| tree construction | XSLT wird auch *tree construction part* von XSL genannt. Eine Transformation produziert eine Baumstruktur, die gegenüber dem Eingabedatenstrom modifiziert sein kann. In die "Blätter" dieses Ausgabebaumes können Texte oder Formatierungsanweisungen eingehängt werden. |
| Differenzen zwischen XML und XSL | Während ein XML-Dokumentbaum aus Element- und Inhaltsknoten besteht, werden in XSL attribute, namespaces, processing instructions und comments ebenfalls als Knoten aufgefasst, die verarbeitet werden müssen. Ein XSL-Stylesheet unterscheidet auch zwischen Document-root und root-Element innerhalb des Dokuments, ignoriert werden DTD und Doctype Definition des Quelldokuments. |
| Schablonen und Regeln | Ein XSL-Dokument enthält eine Reihe von *templates* (Schablonen) und zugehörigen Regeln (rules). Jede *template* Regel definiert ein Muster (pattern), das einem Teilbaum des XML-Dokuments entspricht. Stößt der *XSL*-Prozessor während der Verarbeitung auf das entsprechende Muster, werden die *template*-Anweisungen angewendet, d.h. ihr Inhalt ausgewertet und ausgegeben. Jedes *template* enthält in der Regel Markup, neue Daten und Daten, die aus dem Quelldokument in das neue Dokument einkopiert werden. |

## 4.2 Wie sieht ein XSL-Dokument aus

Ein XSL-Dokument beginnt mit einem Vorspann, der XML-*version* und *encoding* enthalten kann.

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

Der Document root ist mit xsl:stylesheet - erlaubt ist auch xsl:transform - gekennzeichnet (XSLT 2.2).

```
<xsl:stylesheet>
   <!-- Inhalt: (top-level-elements) -->
</xsl:stylesheet>
```

top-level-elements

Innerhalb des Stylesheet-*tags* sind eine Anzahl von festgelegten XSL-Anweisungen in Form von XML-*tags* erlaubt. Alle direkten Nachfahren des Stylesheet Knotens werden `top-level-elements` genannt. Folgende sind in XSLT 2.2 genannt:

- `xsl:import`
- `xsl:include`
- `xsl:strip-space`
- `xsl:preserve-space`
- `xsl:output`
- `xsl:key`
- `xsl:decimal-format`
- `xsl:attribute-set`
- `xsl:variable`
- `xsl:param`
- `xsl:template`

### *4.2.1 templates*

Am häufigsten benutzt ist `xsl:template`, denn jede Regel, die auf das Eingabedokument angewendet werden soll, ist als `match`-Attribut in einer `xsl:template` Anweisung zu formulieren. Die `match`-Regel wird in Form eines Musters (pattern) codiert (XSLT 5).

Die Muster zur Adressierung von Teilbäumen des Quelldokuments wiederum, sind in XPath definiert. Als Beispiel soll der Versuch dienen `postkarte.xml` in Form von HTML sichtbar zu machen.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
                xmlns="http://www.w3.org/TR/xhtml1">
<xsl:template match="/">
   <html>
      <head>
         <META HTTP-EQUIV="Content-Type"
                CONTENT="text/html; charset=iso-8859-1"/>
         <TITLE>HTML Postkarte</TITLE>
      </head>
      <body>
         <xsl:apply-templates />
      </body>
   </html>
</xsl:template>
</xsl:stylesheet>
```

(postkarte0.xsl)

stylesheet-root

Die äußere Klammer wird durch das `<xsl:stylesheet>` root-Element gebildet, dann folgt mit `<xsl:template match="/">` die erste *template*-Regel. Diese Regel fordert den XSLT Prozessor auf, im Quelldokument einen Knoten zu finden, der der Regel `"/"` entspricht, womit Dokument-Root bezeichnet ist.

apply-templates

Ist der Einstieg in das Dokument gefunden, werden die HTML-*tags*, die im Stylesheet eingegeben wurden, in den Ausgabedatenstrom gestellt. Im Beispiel ist das nicht mehr als ein sehr einfaches Gerüst einer HTML Datei.

Mehr würde nicht passieren, wenn nicht die Aufforderung weitere *template*-Regeln anzuwenden mit `<xsl:apply-templates/>` gegeben würde. Diese Anweisung sorgt dafür, daß die Knoten des XML-Dokuments rekursiv durchsucht werden. Bei jedem Knoten wird geprüft, ob es im Stylesheet eine Regel gibt, die auf den Knoten angewendet werden muß. Da wir keine weiteren Regeln in unserem Stylesheet haben, ist es verwunderlich, daß überhaupt ein Ausgabeergebnis sichtbar ist - auch wenn die Darstellung des gesamten Inhalts als fortlaufende Zeichenkette nicht gerade als informativ eingestuft werden kann.

Daß überhaupt irgend etwas sichtbar ist, liegt daran, daß für XSLT Standard *template*-Regeln definiert wurden, die angewendet werden, wenn keine passende Regel im aktuellen Stylesheet auffindbar ist. Sie geben einfach alle Inhalte von Textknoten und Attributen aus.

apply-templates select=" "

Um die Ergebnismenge zu steuern, kann bei `<xsl:apply-templates/>` ein Auswahlkriterium angegeben werden. Soll nur der Text der Postkarte sichtbar werden, wird lediglich
`<xsl:apply-templates select="postkarte/text" />`
angegeben (postkarte1.xsl). Da keine weiteren Angaben über die weitere Verarbeitung gemacht wurden, sorgen wiederum die Standard-*templates* für eine unstrukturierte Ausgabe der verschiedenen Untergliederungen des Text-Knoten.

Durch kleine Modifikationen am Stylesheet läßt sich das ändern:

```
.........
     <body>
       <xsl:apply-templates select="postkarte/text"/>
       <xsl:apply-templates select="postkarte/mfg"/>
     </body>
........
<xsl:template match="gruss">
   <h2> <xsl:apply-templates/> </h2>
</xsl:template>

<xsl:template match="mfg">
   <h2> <i> <xsl:apply-templates/> </i></h2>
</xsl:template>
.......
```

(Auszug aus postkarte2.xsl)

Immerhin sieht das Ergebnis deutlich lesbarer aus. Die beiden neuen *templates* werden in der Reihenfolge des apply-templates Aufrufs ausgeführt.

**Liebe Freunde,**
uns geht es gut, das Wetter und das Essen sind prima, nur das Wasser könnte etwas wärmer sein. Egal, den Kindern gefällt der viele Sand und wir haben unsere Ruhe.
*Viele liebe Grüsse von Euren*

(Umwandlungsergebnis von postkarte2.xsl)

Eine wichtige Fähigkeit von XSLT ist es, die Ausgabereihenfolge frei steuern zu können. Teilbäume des Quelldokuments können beliebig oft, in beliebiger Reihenfolge im Ausgabedokument erscheinen.

Reorganisation der Quelldaten

Bei einer echten Postkarte erscheint die Anschrift auf der rechten Seite, neben der Mitteilung. Auf der Rückseite ist heutzutage meist ein Foto.
Die Plazierung der Anschrift rechts neben der Mitteilung fordert in HTML die Verwendung einer Tabelle. Der nächste Schritt ist demnach, durch Einträge im Stylesheet eine Tabelle generieren zu lassen und die Textteile darin zu positionieren. Der Aufruf der *template*-Regeln könnte so aussehen:

```
<table>
   <tr>
      <td width="342">
         <xsl:apply-templates select="text"/>
         <xsl:apply-templates select="mfg"/>
      </td>
      <td width="262">
         <xsl:apply-templates select="anschrift"/>
      </td>
   </tr>
</table>
```

(Steuerung der Aufrufreihenfolge durch sequentielle Anordnung)

### *4.2.2 Generieren von Text*

Neu kommt das *template* für "anschrift" hinzu, das die Bedeutung eines weiteren XSLT-Elements illustriert: `<xsl:text>` (XSLT 7.2).

```
<xsl:template match="anschrift">
   <blockquote>
      <p>
         <font color="#0000FF">
            <xsl:value-of select="adressat"/>
            <br/>
            <xsl:value-of select="strasse"/>
            <br/>
            <xsl:value-of select="plz"/>
            <xsl:text> </xsl:text>
            <xsl:value-of select="ort"/>
            <br/>
            <xsl:value-of select="land"/>
         </font>
      </p>
   </blockquote>
</xsl:template>
```

("Anschrift" template rule)

whitespace - Problem

Zwischen Postleitzahl und Ortsnamen braucht ein menschlicher Leser ein Leerzeichen. Laut XML-Spezifikation sind Leerzeichen im XML-Code nicht signifikant, d.h. sie werden verschluckt. Durch Angabe eines Leerzeichens innerhalb eines `<xsl:text>` -Elements kann der notwendige Abstand erzeugt werden.

| Hinweis: |
| --- |
| Das `<xsl:text>` -*tag* wird auch verwendet, um *tags* aus dem XSL-Namespace, also `<xsl:...>` -Elemente in ein Ausgabedokument einzutragen. Auf diese Weise ist es möglich, Stylesheets durch Stylesheets erzeugen zu lassen. Die Angabe der öffnenden spitzen Klammer muß dabei mit `&lt;` Ampersant mit `&amp;` codiert werden. |

### *4.2.3 Zusammensetzen von Stylesheets*

xsl:include und xsl:import

Das Anschrift-*template* kann irgendwo im Stylesheet untergebracht werden. Bei komplexen Dokumenten entstehen so leicht lange und unübersichtliche XSL-Dateien. Viele *template*-Regeln - z.B. für Bilder, etc. - braucht man zudem in verschiedenen Zusammenhängen und damit in verschiedenen Stylesheets immer wieder. Es liegt nahe, solche *template*-Regeln in separate Dateien auszulagern und wiederzuverwenden.
XSLT definiert zwei Formen, um Stylesheets zu kombinieren `<xsl:include>` (XSLT 2.6.1) und `<xsl:import>`(XSLT 2.6.1).

Für unser Beispiel soll das Anschrift *template* mit dem `<xsl:stylesheet>`-*tag* umkleidet und unter "`anschrift.xsl`" abgelegt werden. Im Stylesheet wird die Regel mit `<xsl:include href="anschrift.xsl"/>` verfügbar gemacht. Während der Verarbeitung wird die im include angegebene Datei geparst. Die Kindknoten des `xsl:stylesheet` Knotens werden an der Stelle des `xsl:include` Statements eingefügt. Eine Kombination von Stylesheets wirkt sich auf die Reihenfolge der Verarbeitung nicht aus. Wichtig ist, daß das `xsl:include` -Statement ein Top-Level Element sein muß. Die Kombination von Stylesheets mit `<xsl:import>` wirkt vergleichbar, mit der Ausnahme, daß die *template*-Regeln des importierenden Stylesheet vor den importierten Vorrang haben.

### 4.2.4 Zugriff auf Attribute

xsl:value-of

Bislang haben wir im Stylesheet lediglich Elemente angesprochen. Das Bild unserer Postkarte wird in `<img src="xanta.jpg"/>` referenziert. Der Dateiname, des Bildes ist als Attribut des img-*tags* codiert. Der Zugriff auf XML-Attribute wird in XSLT, genauer in XPATH mit @ eingeleitet. Im Postkartenbeispiel wird das XML img-*tag* als HTML img-*tag* formatiert. Als "Zugriffsmethode" wird explizit der Wert des select Ausdrucks abgefragt:
**`<xsl:value-of select="@src"/>`**
Dieser Ausdruck wird bei der Verarbeitung durch den Wert aus dem XML-Dokument (xanta.jpg) ersetzt. Im Unterschied zu `apply-templates`, wird mit `value-of` keine weitere Rekursion angestoßen, sondern lediglich der Wert des Ausdrucks zurückgeliefert. Weil es sich im Beispiel um einen berechneten und nicht um einen fixen Wert handelt, ist zur Kodierung des HTML-IMG *tags* keine Anweisung im Sinne von
`<img border="1" src="xanta.jpg">` möglich. Das SRC-Attribut im HTML-*tag* muß mithilfe einer <xsl:attribute> Anweisung erzeugt werden. Folgendes *template*

Wert eines Attributs lesen

```
<xsl:template match="img">
     <img border="1">
       <xsl:attribute name="src">
         <xsl:value-of select="@src"/>
       </xsl:attribute>
     </img>
  </xsl:template>
```

erzeugt diesen HTML - Code:

```
<img border="1" src="xanta.jpg"/>
```

### 4.2.5 XPath Navigation im XML-Baum

location path

Die Postkarte ist fast fertig, das Datum aus dem XML-Dokument fehlt noch. Es wird mit einem XPath-Ausdruck aus dem Dokument-Element geholt. Der aktuelle Knoten ist ein untergeordneter Knoten des Dokumentknotens. Im Beispiel greifen wir mit einer absoluten Angabe der Lokation (location path) auf ein Attribut des Dokument-Root Knotens zu.

```
<xsl:template match="mfg">
   <h2><i><xsl:apply-templates/></i></h2>
   <xsl:value-of select="/postkarte/@datum"/>
</xsl:template>
```

Elementtypen in XPath

XPath unterscheidet im XML-Dokument zwischen Elementen (element node), Attributen (attribute node) und Text (text node). XPath-Ausdrücke können eine Lokationsinformation oder eine Funktionsinformation beschreiben. Am häufigsten sind die Lokationsinformationen (location paths). Folgende Beispiele illustrieren die Zugriffsmöglichkeiten auf Dokumententeile mithilfe von XPath-Anweisungen.

| XPath-Ausdruck | Bedeutung |
|---|---|
| `child::para[position()=1]` | Erstes `para` Kindelement des Kontextknotens |
| `child::para[position()=last()]` | Letztes `para` Kindelement des Kontextknotens |
| `child::para[position()=last()-1]` | Vorletztes `para` Kindelement des Kontextknotens |
| `child::para[position()>1]` | Alle ausser dem ersten .... |
| `following-sibling::chapter[position()=1]` | erster folgender `chapter` Geschwisterknoten des Kontexts |
| `preceding-sibling::chapter[position()=1]` | erster vorausgehender `chapter` Geschwisterknoten des Kontexts..... |

(XPath-Ausdrücke - explizite Schreibweise)

Für viele, häufige Anwendungsfälle ist eine gekürzte Schreibweise erlaubt, die leichter zu merken ist:

| | |
|---|---|
| `para` | Alle `para` Kindelemente des Kontexts |
| `@name` | Das `name` Attribut des Kontexts |
| `@*` | Alle Attribute des Kontexts |
| `para[1]` | Erstes `para` Kindelement des Kontexts |
| `para[last()]` | Das letzte `para` Kindelement des Kontexts |
| `/doc/chapter[5]/section[2]` | Das zweite `section` Element des fünften `chapter` Elements von `doc` |
| `//para` | Alle `para` Nachkommen des document root |
| `.` | Auswahl des Kontextknotens |
| `....` | |

(XPath-Ausdrücke - Kurzschreibweise)

Daneben definiert XPath noch Funktionen, wie z.B. last(), first() etc. Die ausführliche Dokumentation zu den Funktionen findet sich in XPath Abschnitt 4. Weitere Details werden in den praktischen Übungen angesprochen.

# 5 Praxisversuche

Durch die praktischen Übungen soll ein persönlicher Steckbrief, d.h. ein webbasiertes Personenprofil entstehen. Der Ablauf der Übungen ist so aufgebaut, daß zunächst eine XML-Struktur erstellt wird, die die persönlichen Daten aufnimmt.
Diese erste Übung führt in den Umgang mit XML-Dateien und in den verwendeten XML-Editor ein.

In einem weiteren Durchgang werden XSL-Stylesheets erzeugt, die es erlauben die erstellten Daten in eine einfache HTML-Seite zu transformieren. Änderungen am Stylesheet fügen der HTML-Seite ein Inhaltsverzeichnis hinzu.

## 5.1 Eine einfache HTML-Seite

### *5.1.1 Erstellen einer XML-Datei*

Folgende DTD definiert ein einfaches Personenprofil. Erstellen Sie eine Datei, die dieser DTD entspricht.

```
<!ELEMENT person (adresse, profil) >
<!ATTLIST person vorname CDATA #IMPLIED>
<!ATTLIST person nachname CDATA #IMPLIED>
<!ATTLIST person geb CDATA #IMPLIED>
<!ELEMENT adresse (telekom)* >
<!ATTLIST adresse str CDATA #IMPLIED>
<!ATTLIST adresse plz CDATA #IMPLIED>
<!ATTLIST adresse ort CDATA #IMPLIED>
<!ELEMENT telekom (art|code)* >
<!ELEMENT art (#PCDATA)* >
<!ELEMENT code (#PCDATA)* >
<!ELEMENT profil (thema)* >
<!ELEMENT thema (item)* >
<!ATTLIST thema id CDATA #IMPLIED>
<!ATTLIST thema titel CDATA #IMPLIED>
<!ATTLIST thema zeitraum CDATA #IMPLIED>
<!ELEMENT item (#PCDATA)* >
```

Nutzen Sie dazu die Schablone "lebenslauf.xml" im Übungsordner und ergänzen Sie die fehlenden *tags* und den Inhalt mit ihren Angaben. Beachten Sie bitte, daß diese Schablone die Referenz auf obige DTD über das Netz definiert. Änderungen an der lokalen DTD machen sich lediglich nach der Änderung dieser Referenz auf diese lokale DTD bemerkbar.

### *5.1.2 Erstellen eines einfachen Stylesheets*

Erstellen Sie ein einfaches Stylesheet, um die Inhalte der XML-Datei sichtbar zu machen. Benutzen Sie dazu die Vorlage "`lebenslauf0.xsl`". Durch diese Vorlage werden Teile des Dokuments sichtbar.

- Erweitern Sie die Vorlage durch Ausprogrammierung der entsprechenden *templates* für <adresse>, <profil>, <thema> und <item>.
- Erweitern Sie danach das *template* für <item> durch die Einblendung einer Grafik als Markierungspunkt vor dem Text. Erzeugen Sie dazu eine weitere Tabellenzelle vor <item>, hierin eine Referenz auf die Grafik "`blue-ball-small.gif`".

Benutzen Sie für den Umwandlungslauf das Skript "`transform.bat`", das sich ebenfalls im Übungsordner befindet. In diesem Skript werden die CLASSPATH Einträge für die Parser- und XSL-Prozessor Bibliotheken gesetzt, und der XSL-Prozessor aufgerufen.
Geben Sie als Aufrufparameter den Namen (ohne Erweiterung) der XML-Datei, dann den Namen der XSL-Datei an (ebenfalls ohne Erweiterung). Wenn die Transformation gelingt, erzeugt das Skript eine HTML-Datei, die den Namen der XML-Datei trägt. Zur Kontrolle wird die Grösse der HTML-Datei am Ende des Skripts ausgegeben.

Aufrufbeispiel: `transform lebenslauf lebenslauf0`

## 5.2 Eine HTML-Seite mit Inhaltsverzeichnis

In dieser Übung soll das Quelldokument zwei maldurchlaufen werden. Ein erster Durchlauf wird dazu benutzt, eine Inhaltsübersicht über alle Themen, bzw. deren titel-Attribute zu erstellen. Der zweite Durchlauf stellt die eigentlichen Inhalte der XML-Datei in lesbarer Form her.

- Kopieren Sie dafür ihr bisher erstelltes Stylesheet auf den Namen `LL_TOC.XSL`.
- Fügen Sie dem `<xsl:apply-templates select="thema">` Aufruf das Attribut `mode="TOC"` hinzu.
- Kopieren sie diesen Aufruf und ersetzen Sie in der neuen Zeile `mode="TOC"` durch `mode="CONT".`
- Duplizieren Sie das template `<xsl:template match="thema">` und fügen Sie entsprechend den obigen Definitionen auch hier jeweils `mode="TOC"` `mode="CONT"` hinzu.
- Verändern Sie die beiden `thema`-*templates* hinsichtlich ihrer Funktion als Inhaltsverzeichnis und als Inhaltsdarstellung entsprechend Ihrer Vorlieben ab.

# 6 Anhang

## 6.1 SAX Beispielprogramm

Das Programm registriert einen einfachen event-handler. Jeder Knotentyp wird lediglich einmal ausgegeben, auch wenn er im Quelldokument mehrfach vorkommt. Beim Durchgang durch die Datei wird für jeden Knoten ein *XSL-template* angelegt. Die Laufzeitumgebung muß über einen CLASSPATH - Eintrag verfügen, der auf einen SAX-fähigen Parser verweist.

```
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.*;
import sax.helpers.AttributeListImpl;
import org.xml.sax.AttributeList;
import org.xml.sax.HandlerBase;
import org.xml.sax.Parser;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.ParserFactory;

/**
* ein einfaches SAX Programm, das ein XSL-Skelett für die Eingabedatei
* erzeugt
* Autor M.Knobloch 1999, basiert auf Beispiel von IBM XML4J
*/

public class NodeLister extends HandlerBase
{
/** Default parser name. */
private static final String DEFAULT_PARSER_NAME =
"com.ibm.xml.parsers.SAXParser";
/** Print writer. */
protected PrintWriter out;
/** hash table for bufferd node names  **/
protected Hashtable ht;

// Constructor
protected NodeLister()
throws UnsupportedEncodingException
{
    ht = new Hashtable();
    out = new PrintWriter(new OutputStreamWriter(System.out));
}

/** Prints the output from the SAX callbacks. */
public static void print(String parserName, String uri)
{
    try {
        HandlerBase handler = new NodeLister();
        Parser parser = ParserFactory.makeParser(parserName);
        parser.setDocumentHandler(handler);
        parser.setErrorHandler(handler);
        parser.parse(uri);
```

```
        }

    catch (Exception e) {
        e.printStackTrace(System.err);
    }
} // print(String,String)

//
// DocumentHandler methods
//
/** Start document. */
public void startDocument()     {
    out.println("<?xml version=\"1.0\" encoding=\"ISO-8859-1\" ?>");
    out.println("<xsl:stylesheet
xmlns:xsl=\"http://www.w3.org/XSL/Transform/1.0\" >");
}

/** End document. */
public void endDocument() {
    out.println("</xsl:stylesheet>");
    out.flush();
}

/** Start element. */
public void startElement(String name, AttributeList attrs)     {
    if (!ht.containsKey(name))
    {
        ht.put(name, name);
        out.println("");
        out.print("<xsl:template match=\"");
        out.print(name);
        out.println("\">");
        out.println("\t<xsl:apply-templates />");
        out.println("</xsl:template>");
    }
}

/** Warning. */
public void warning(SAXParseException ex) {
    System.err.println("[Warning] "+ ex.getMessage());
    }

/** Error. */
public void error(SAXParseException ex) {
    System.err.println("[Error] "+ ex.getMessage());
    }

/** Fatal error. */
public void fatalError(SAXParseException ex) throws SAXException {
    System.err.println("[Fatal Error] "+ ex.getMessage());
    throw ex;
    }
//
// Main
//
public static void main(String argv[]) {
    if (argv.length == 0)
    {
        System.err.println("usage: java sax.NodeLister uri ...");
        System.err.println();
```

```
        System.exit(1);
    }
    // vars
    String  parserName = DEFAULT_PARSER_NAME;
    String arg = argv[0];

    // print uri
    System.err.println(arg+':');
    print(parserName, arg);
    System.out.println();
}
} // class NodeLister
```

## 6.2 postkarte.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
xmlns="http://www.w3.org/TR/xhtml1">

<!--
Postkarte-template als Einstieg in das Dokument
-->

<xsl:template match="postkarte">
     <html><head>
          <META HTTP-EQUIV="Content-Type" CONTENT="text/html;
                                          charset=iso-8859-1"/>
          <TITLE>HTML Postkarte</TITLE>
        </head>
        <body>
          <table>
            <tr>
              <td width="608">
                <xsl:apply-templates select="img"/>
              </td>
            </tr>
          </table>
          <table>
            <tr>
              <td width="342">
                <xsl:apply-templates select="text"/>
                <xsl:apply-templates select="mfg"/>
              </td>
              <td width="262">
                <xsl:apply-templates select="anschrift"/>
              </td>
            </tr>
          </table>
        </body>
     </html>
</xsl:template>
```

```
<!--
Anschrift-template
-->
<xsl:template match="anschrift">
   <blockquote>
      <p>------------------</p>
      <p>Marke aufkleben!</p>
      <p>------------------</p>
   </blockquote>
   <blockquote>
          <xsl:value-of select="adressat"/><br/>
          <xsl:value-of select="strasse"/><br/>
          <xsl:value-of select="plz"/>
          <xsl:text disable-output-escaping="yes"> </xsl:text>
          <xsl:value-of select="ort"/>
          <br/>
          <xsl:value-of select="land"/>
   </blockquote>
</xsl:template>

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

<xsl:template match="gruss">
   <h2><xsl:apply-templates/></h2>
</xsl:template>
<!--
        Gruss und nachfolgendes Datum mit XPATH Anweisung
         aus current node mfg auf /postkarte/@datum
-->

   <xsl:template match="mfg">
      <h2><i><xsl:apply-templates/></i></h2>
      <xsl:value-of select="/postkarte/@datum"/>
   </xsl:template>

<!--
IMG-template, sollte ausgelagert werden und flexibler definiert sein
-->

<xsl:template match="img">
   <img border="1">
      <xsl:attribute name="src"><xsl:value-of select="@src"/>
      </xsl:attribute>
   </img>
</xsl:template>

</xsl:stylesheet>
```

## 6.3 Links

**XML-Spezifikationen**:

http://www.w3.org/TR/1998/REC-xml-19980210
Das normative englische Original.

http://www.mintert.com/xml/trans/REC-xml-19980210-de.html
Deutsche Fassung der Spezifikation (auch i. Buch von Mintert/Behme ).

http://www.w3.org/TR/xmlschema-1/
Der Working Draft zu XML Schema Part 1: Structures.

http://www.w3.org/TR/xmlschema-2/
Der Working Draft zu XML Schema Part 2: Datatypes.

http://www.w3.org/TR/xml-stylesheet/
Die Recommandation zur Verbindung Verknüpfung von XML-Dokumenten mit Stylesheets per Processing Instruction.

**XSL-Spezifikation:**

http://www.w3.org/TR/WD-xslt
Die Recommandation zum Teil I, der XSL Transformation Language.

http://www.w3.org/TR/xpath/
Die Proposed Recommandation zur Navigation im XML-Quelldokument.

http://www.w3.org/TR/WD-xsl/
Der Working Draft zum Teil II, der XSL Formatierungs Language.

**XML-Software, Anwendungen, Projekte, links, samples**

http://www.xmlsoftware.com
Vorbildlich kategorisierte und kommentierte Sammlung von XML und verwandten Technologien, die regelmäßig gepflegt wird.

http://www.alphaWorks.ibm.com/Home/
Die neuen Technologien und Tools, die IBM auf der Basis von XML bzw. XML und Java anbietet.

http://www.oasis-open.org/cover/xml.html#applications
Kurzbeschreibungen und – sofern vorhanden – Links zu bereits realisierten oder in Arbeit befindlichen Software-Lösungen mit XML.

http://www.xml.org
XML-Portal für die Industrie.

http://xml.apache.org
Apaches XML-Engangement ist hier vollständig vertreten. Eingegliedert sind inzwischen IBM's XML4J Parser (jetzt XERCES), LotusXSL (jetzt XALAN) und James Taubers Formatting objects for PDF (FOP).

http://members.aol.com/xmldoku
Eine kurze gut verständliche Einführung in XML.

http://msdn.microsoft.com/xml/tutorial/default.asp
Interaktiver Workshop mit mehreren Lektionen.

http://msdn.microsoft.com/workshop/xml/index.asp
Workshop für XML Entwickler mit guter Inhaltsübersicht.

http://metalab.unc.edu/xml/books/bible/examples/
Eine Menge XML/XSL/CSS-Dateien zur sogenannten XML Bible von Elliotte Rusty Harold (s. Literaturliste). Wohl nur mit Buch oder Vorkenntnissen sinnvoll.

http://www.mulberrytech.com/xsl/xsl-list
Liste mit regem Verkehr zu Themen sehr unterschiedlicher Richtung. Auch die Profis diskutieren heftig mit.

http://www.heise.de/ix/raven/Web/xml/default.html
Der Heise Verlag unterhält im Rahmen der Zeitschrift iX eine XML-Site mit vielen nützlichen Links zu Projekten, Konferenzen und FAQs.

http://www.WDVL.com/Authoring/Languages/XML/Resources.html
Riesige Sammlung zu so ziemlich allen Themen rund um XML.

http://metalab.unc.edu/xml/
Elliotte Rusty Harold's Site bietet so ziemlich alles und immer das Neueste zum Thema XML. Wichtig sind vor allem Konferenzen und News.

http://www.xml.com
Ein Partnerprojekt von Seybold Publications und Songline Studios, das zu O'Reilly & Associates gehört. Viele gute Artikel von den ‚Mitautoren' von XML wie z.B. Tim Bray, Lisa Rein, Norman Walsh oder James Clark.

## 6.4 Literatur

**Teach yourself XML in 21 days:** North, Simon, 1999, ISBN: 1575213966, Sams

**XML Bible**: Eliotte Rusty Harold, 1999, ISBN: 0764532367, IDG Books Worldwide Inc; derzeit n. lieferbar. Auszüge im Web verfügbar.