

Die eXtensible Stylesheet Language

Teleseminar XML: Anfrage- und Transformationssprachen
Uni Karlsruhe/Uni Freiburg

von Christian Gieche & Florian Weitzling

1. Einführung	3
1.1. Hintergrund	3
1.1.1. Trennung von Inhalt und Darstellung.....	3
1.2. Der Ursprung von XSL	3
1.3. Was ist XSL?.....	4
2. XSL-T	4
2.1. Wie funktioniert XSLT?	4
2.2. XPath.....	7
2.3. XSLT Befehle	8
2.3.1. <xsl:template>	9
2.3.2. <xsl:apply-templates>	9
2.3.3. <xsl:value-of>.....	10
2.3.4. weitere XSL-Befehle	10
2.4. Zwei XSLT-Prozessoren im Vergleich.....	11
2.4.1. Xalan.....	11
2.4.2. Start von Xalan als Standalone-Programm.....	11
2.4.3. Start von Xalan per API-Aufruf	12
2.4.4. Internet Explorer.....	12
2.4.5. Das Standardverfahren	14
2.4.6. Das Verfahren des Internet Explorers bei eingebetter processing instruction.....	15
2.4.7. Xalan und Internet Explorer in der Übersicht	16
3. XSL-FO	16
3.1. Was ist XSL-FO?	16
3.2. Praxis.....	16
3.3. Vergleich XSL-FO vs. CSS	18
3.4. Fazit.....	18
Anhang A: Quellen	20

1.Einführung

1.1. Hintergrund

Die Datenbeschreibungssprache XML erfreut sich steigender Beliebtheit. Ihre Akzeptanz und somit auch ihre Verbreitung wächst mit steigender Geschwindigkeit.

Schaut man sich aber einmal ein XML Dokument an, so ist es zwar lesbar, die Darstellung als Quelldokument an sich läßt aber einiges zu wünschen übrig. Zunächst stören die Auszeichnungen ein wenig, nimmt man diese aber weg, so gehen Informationen und die ganze Struktur verloren.

Diese relative Unlesbarkeit resultiert daraus, daß bei XML der Inhalt und die Darstellung strikt getrennt sind. XML selbst beinhaltet im Gegensatz zum XML-Vokabular HTML keine Formatierungselemente, sodaß ein Browser von einem XML Dokument mehr oder weniger (der Internet Explorer benutzt eine „auf- und zusammenklappbare“ Baumstruktur) nur den Quelltext darstellt.

1.1.1. Trennung von Inhalt und Darstellung

Die grundsätzliche Trennung von Inhalt und Darstellung bietet ein Reihe von Vorteilen. Durch die Unabhängigkeit des Inhalts von der Darstellung kann die Darstellung flexibel an die Art der Ausgabe angepaßt werden.

So ist es z.B. möglich, dieselben Daten in angemessener Form etwa sowohl auf dem Display eines Mobiltelefons als auch auf dem Monitor eines PCs auszugeben. Weitergehend ist es sogar denkbar, dasselbe Dokument sowohl für die Text- als auch für die Sprachausgabe zu formatieren.

Außer der Art des Ausgabemediums ist es auch möglich, den Umfang der Darstellung des Dokumentes zu variieren. Aus dem selben Dokument kann dann sowohl eine komplette Ausarbeitung als auch eine Zusammenfassung oder nur ein Inhaltsverzeichnis erstellen.

Neben der Struktur des Dokuments kann sich die Verarbeitung zur Darstellung eines Dokuments auch auf das visuelle Erscheinungsbild beschränken. Als Anwendungsbeispiele sind hier etwa die Corporate Identities von Firmen zu nennen, die dann beliebige Dokumente mit einem Standardlayout versehen können.

1.2. Der Ursprung von XSL

Die erste Möglichkeit zur Formatierung von XML Dokumenten wurde Ende 1996 mit CSS (Cascading Style Sheets) vorgestellt. CSS ist in seinen Möglichkeiten allerdings recht limitiert und beschränkt sich im Gegensatz zu XSL (wie wir später sehen werden) auf die visuelle Formatierung von XML Dokumenten.

CSS bietet aber eine angemessene Möglichkeit, die Vermischung von Inhalt und Darstellung von HTML-Dokumenten aufzuheben und bietet sich daher als ideale (und mittlerweile auch einigermaßen akzeptierte) Ergänzung zu HTML an.

Aus dieser Situation heraus wurde XSL entwickelt. Unter dem Dach des World Wide Web Consortium (W3C) wurde unter Mitarbeit von Firmen wie z.B. Microsoft oder ArborText im August 1997 ein Vorschlag für XSL erstellt.

1.3. Was ist XSL?

XSL besteht aus zwei Teilen:

- Der Transformationssprache XSLT, die ein XML Dokument in ein anderes transformiert, wobei sowohl Hinzufügen, Entfernen, als auch Ändern von Elementen möglich ist
- Der Formatierungssprache XSL-FO, die XML Dokumente für die Ausgabe formatiert, in der Funktion ungefähr vergleichbar mit CSS, aber wesentlich mächtiger

2. XSL-T

2.1. Wie funktioniert XSLT?

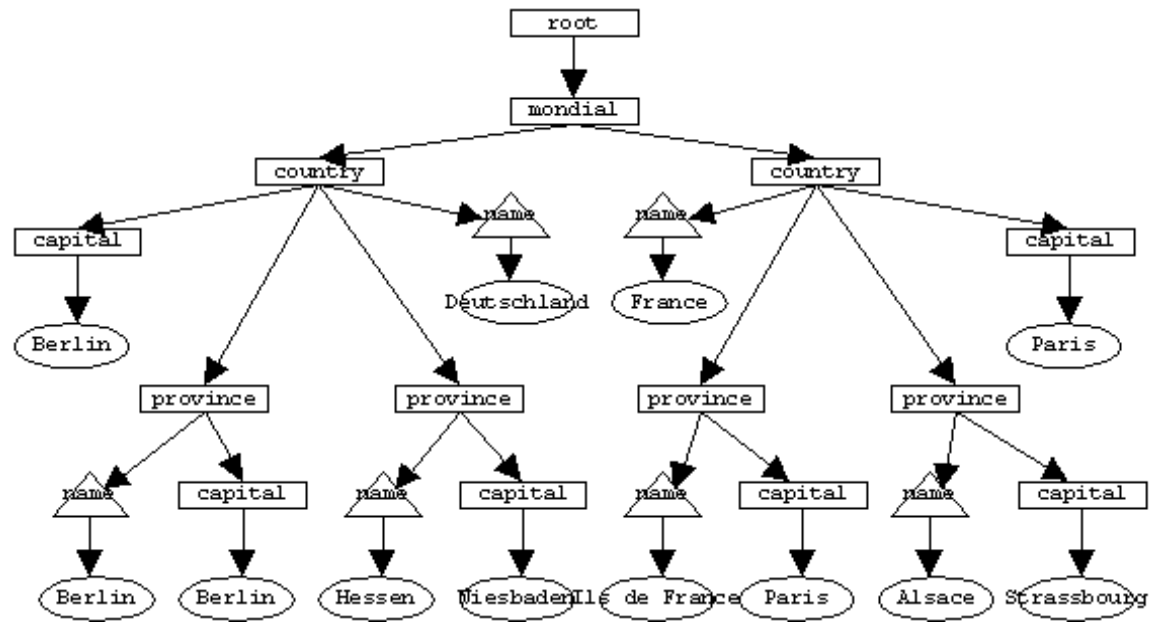
Der Ausgangspunkt für eine XSLT-Transformation sind das XML-Quelldokument und das XSL-Stylesheet, welches die Regeln für die Transformation enthält. Das XSL Stylesheet selbst ist wiederum ein XML Dokument, da die Befehle von XSLT der XML- Syntax folgen.

Das Quelldokument wird zunächst von einem XML Parser in einen sogenannten „source tree“ umgewandelt. Dies ist eine baumartige Struktur, die den Aufbau des XML-Dokumentes widerspiegelt. Die Elemente dieser Baumstruktur heißen „nodes“. Bei der Umwandlung der XML-Datei in den source tree werden alle Arten von Elementen zu nodes. Dazu zählen z.B. die Auszeichnungselemente, deren Attribute und auch normaler Text. Die Wurzel „root“ dieser Baumstruktur ist allerdings nicht das umschließende Element des XML-Dokuments, sondern wird vom Parser als Wurzel die sogenannte „root node“ hinzugefügt.

Zur Verdeutlichung des Vorgangs bei der Umwandlung der XML-Datei in einen source tree hier ein Beispiel mit Quelldatei und generiertem source tree:

```
<? xml version="1.0" encoding="ISO-8859-1" ?>
<mondial>
  <country name="Germany">
    <capital>Berlin</capital>
    <province name="Berlin">
      <capital>Berlin</capital>
    </province>
    <province name="Hessen">
      <capital>wiesbaden</capital>
    </province>
  </country>
  <country name="France">
    <capital>Paris</capital>
    <province name="Alsace">
      <capital>Strassbourg</capital>
    </province>
    <province name="Ils de France">
      <capital>Paris</capital>
    </province>
  </country>
</mondial>
```

Aus dieser Datei wird die folgende Baumstruktur generiert, mit der XSLT dann weiter arbeitet:

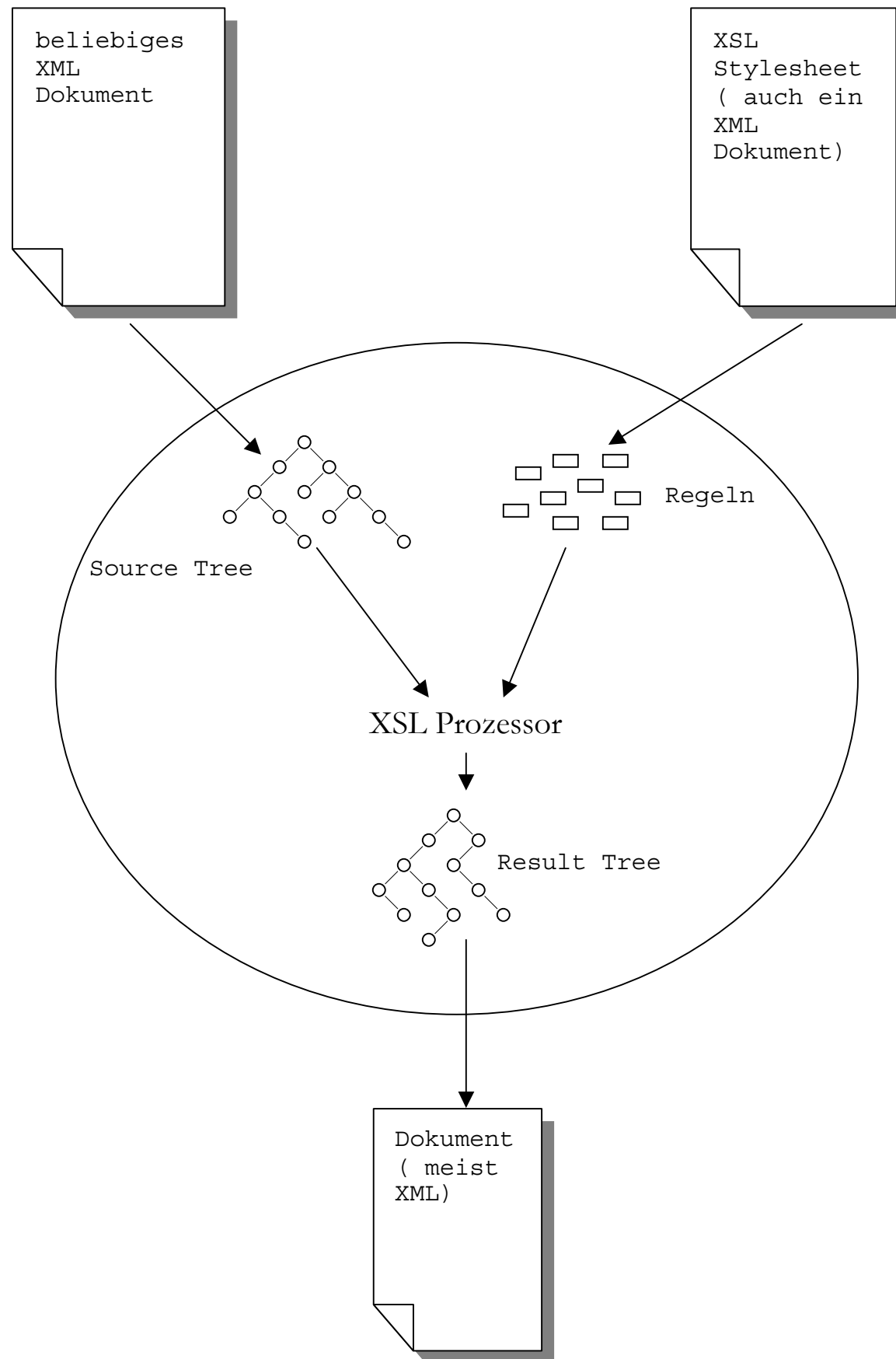


Dieser source tree wird nun an den XSLT-Prozessor weitergegeben. Dieser wendet nun die Regeln, die er aus dem XSL Stylesheet erstellt hat, auf diesen source tree an und erstellt damit als Ausgabe den „result tree“.

Dieser result tree ist der durch die XSLT-Regeln veränderte source tree, wobei dessen Elemente beliebig verarbeitet, neue Elemente hinzugefügt oder einige herausgenommen worden sein können.

Der result tree kann dann je nach Ausgabeart weiter verarbeitet werden, der Standard ist hier wohl die Serialisation in eine XML-Datei.

In der nachfolgenden Abbildung ist der gesamte Vorgang der XSLT-Verarbeitung noch einmal grafisch dargestellt:



Schema der Verarbeitung von XML Dokumenten mittels XSLT

2.2. XPath

Da eine Regel aus dem XSL-Stylesheet immer auf ein bestimmtes Element angewendet wird, benutzt XSLT den Auswahlmechanismus von XPath, um das jeweilige Element zu bestimmen.

Die Notation von XPath übernimmt dabei Elemente der Notation eines Dateisystems. Wie beim Dateisystem auch wird das Zielelement relativ zum aktuellen Pfad angegeben. So wählt beispielsweise „.“ das aktuelle Element aus, mit „/“ wird in der Hierarchie nach unten gegangen, „/“ alleine wählt die Wurzel der Baumstruktur aus. Auch aus dem Dateisystem bekannt ist das wildcard-Zeichen „*“ mit dem ein Element beliebigen Namens ausgewählt werden kann. Als Erweiterung kennt XPath das „//“ womit ein Element auf einer beliebigen Anzahl von Hierarchiestufen tiefer ausgewählt werden kann, d.h. ein Element das sich irgendwo in dem Sub-Baum befindet, von dem das momentane Element die Wurzel bildet.

Mit dem „oder“-Operator („|“) kann man mehrere Bedingungen zur Auswahl eines Elementes verknüpfen.

Natürlich bietet XPath auch die Möglichkeit, nicht das Element selbst, sondern das Attribut eines Elementes auszuwählen. Dies geschieht einfach durch die Verwendung von „@“, welches die Auswahl eines Attributs kennzeichnet.

Im Folgenden ein paar Beispiele auf Basis dieses Ausschnitts des Beispieldokuments, das schon in Abschnitt 2.1. verwendet wurde:

```
...
<country name="Germany">
  <capital>Berlin</capital>
  <province name="Berlin">
    <capital>Berlin</capital>
  </province>
  <province name="Hessen">
    <capital>wiesbaden</capital>
  </province>
</country>
...
```

Wenn „<country name="Germany">“ das aktuelle Element ist, dann wählt

- „.“
 <country name="Germany">
 ...
 </country>
aus
- „./province/@name“
 name="Berlin"
 name="Hessen"
aus
- „//capital“
 <capital>Berlin</capital> (Unterelement von "country")

```
<capital>Berlin</capital> (Unterelement von "province")
<capital>wiesbaden</capital>
```

aus

- „./capital | ./province/capital“ auch

```
<capital>Berlin</capital> (Unterelement von "country")
<capital>Berlin</capital> (Unterelement von "province")
<capital>wiesbaden</capital>
```

aus

- „./province/*“

```
<capital>Berlin</capital> (Unterelement von "province")
<capital>wiesbaden</capital>
```

aus

Weiterhin bietet XPath die Möglichkeit, das ausgewählte Element genauer zu spezifizieren. Mit „[]“ lassen sich Auswahlkriterien angeben. Als Möglichkeiten bieten sich sowohl der Vergleich auf den Inhalt des Elements, als auch die Auswahl aufgrund der Reihenfolge an:

mit „./province[@name='Hessen']“ wird

```
<province name="Hessen">
</province>
```

ausgewählt

während mit „./province[1]“

```
<province name="Berlin">
</province>
```

ausgewählt wird.

XPath bietet noch weitere Möglichkeiten der Auswahl. So kann mit der id()-Funktion speziell anhand des XML-Attributtyps „id“ selektiert werden, oder mit „processing-instruction()“ können explizit die XML-Processing-Instructions selektiert werden.

2.3. XSLT Befehle

Wie bereits weiter oben erwähnt, ist ein XSL-Stylesheet selbst wieder ein XML-Dokument. Die XSL Befehle richten sich somit nach dem XML-Syntax. Für die XSL-Befehle existiert ein eigener namespace, der mit dem prefix „xsl:“ gekennzeichnet wird.

Zur Orientierung vorweg ein Beispiel für ein simples XSL-Stylesheet, das Daten eines XML-Dokumentes in einer HTML-Datei ausgibt:

```
<?xml version='1.0'?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
```

```

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

<xsl:template match="country">
  <p><xsl:value-of select="@name"/></p>
</xsl:template>

<xsl:template match="province/capital">
  <i><xsl:value-of select="."/></i>
</xsl:template>

</xsl:stylesheet>

```

Angewendet auf die Beispieldatei aus Abschnitt 2.1. ergibt sich die folgende Ausgabe:

```

<?xml version='1.0'?>
<html>
<p>
  Germany
  <i>Berlin</i>
  <i>Frankfurt</i>
</p>
<p>
  France
  <i>Strassbourg</i>
  <i>Paris</i>
</p>
</html>

```

2.3.1. <xsl:template>

<xsl:template> definiert die bereits weiter oben erwähnten Regeln des XSL-Stylesheets.

Zunächst wählt es mit dem Attribut „match“ das/die zu verarbeitende(n) Element(e) mittels XPath aus. Der Inhalt von <xsl:template> definiert dann den Teil des result trees, der für die mittels „match“ ausgewählten Elemente des source trees ausgewählt werden.

Sollte es mehrere Regeln geben, die auf ein Element zutreffen (z.B. wird das Element „province“ sowohl von „province“ als auch von „*“ gematched), so wird die präzisere Auswahl verwendet (in diesem Fall also „province“).

2.3.2. <xsl:apply-templates>

Nachdem der source tree und die Regeln erstellt wurden, versucht der XSLT Prozessor nun, die nodes des source trees mit den Regeln zu „matchen“. Er beginnt dabei mit der root-node und arbeitet sich dann weiter nach unten durch den source tree durch. Dabei fährt er aber nur weiter nach unten, wenn er in der aktuellen Regel den Befehl <xsl:apply-templates> vorfindet, der ihn anweist, die Elemente des sub-trees des aktuellen Elementes auf passende Regeln hin zu untersuchen. Da dieser Prozeß ohne eine Regel für die root-node folglicherweise gar nicht in Gang kommen würde, ist im XSL-Prozessor eine „voreingestellte“ Regel vorhanden, die die root-node auswählt und die Verarbeitung in Gang setzt:

```

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

```

Wie zu erkennen ist, hält diese Standardregel den Prozeß auch „in Gang“, indem einfach mittels „*“ für alle Elemente standardmäßig der sub-tree verarbeitet wird. Diese Standardregel kann natürlich auch überschrieben werden, indem man einfach im XSL-Stylesheet explizit eine Regel für die root-node angibt. Die Standardregel für „*“ wird einfach mit jeder präziseren Auswahl überschrieben.

Während `<xsl:apply-templates>` einfach alle Elemente des sub-trees weiter verarbeitet, kann man mit dem Attribut „select“ die Auswahl der Elemente einschränken. Auch hier wird wieder die XPath Notation benutzt.

2.3.3. `<xsl:value-of>`

In jeder Regel kann der statische Inhalt für die Ausgabe des result trees direkt im Stylesheet angegeben werden. Der dynamische Inhalt, der abhängig von den ausgewählten Elementen des source trees erzeugt wird, wird mit `<xsl:value-of>` generiert.

Generell berechnet `<xsl:value-of>` den Wert eines Ausdrucks, der im Attribut „select“ mitgegeben wird. So ist ein simpler Fall denkbar, indem einfach mittels `<xsl:value-of select="3+4" />` der Wert „7“ ausgegeben wird. Üblicherweise wird im „select“-Attribut aber das Element per XPath spezifiziert, dessen Wert dann ausgegeben wird.

Auch hier gibt es in jedem XSL-Prozessor eine Standardregel, die einem das Leben erleichtert:

```
<xsl:template match="text()|@">
  <xsl:value-of select="."/>
</xsl:template>
```

Mit „text()“ werden alle Elementinhalte und mit „@“ sämtliche Attributwerte gematched, deren Werte dann direkt ausgegeben werden. Sollte also die Verarbeitung mittels `<xsl:apply-templates>` bis zu diesen Elementen gelangen, werden sie, solange keine präzisere Regel existiert, direkt ausgegeben.

2.3.4. weitere XSL-Befehle

XSL beinhaltet auch Konstrukte wie sie ähnlich in gängigen Programmiersprachen wie z.B. Java vorkommen.

So kann man mittels `<xsl:for-each>` eine Art Schleife erzeugen, die nacheinander alle mittels „select“ ausgewählten nodes verarbeitet:

```
<xsl:for-each select="province">
  <xsl:value-of select="."/>
</xsl:for-each>
```

Mit `<xsl:if>` und `<xsl:choose>/<xsl:when>` lassen sich Konditionalabfragen realisieren (Pendants in Java sind „if“ und „switch/case“):

```
<xsl:if test="$somecondition">
  .
</xsl:if>
<xsl:choose>
```

```

    <xsl:when test="$count == 1"> ... </xsl:when>
    <xsl:when test="$count == 2"> ... </xsl:when>
    <xsl:otherwise> ... </xsl:otherwise>
</xsl:choose>

```

Auch die Möglichkeit zur Variablendefinition ist in XSL vorhanden. Mittels `<xsl:variable>` wird eine Variable definiert, auf die dann mit dem „\$“ Operator zugegriffen wird (siehe Beispiele zu `<xsl:if>` und `<xsl:choose>`):

```

<xsl:variable name="continent">
  Europa
</xsl:variable>

```

Der result tree läßt sich auch direkt modifizieren, indem man mit Hilfe von `<xsl:element>`, `<xsl:attribute>` und `<xsl:copy>` Elemente und Attribute selbst erzeugen, bzw. bereits vorhandene kopieren und diese dann direkt in den Ergebnisbaum einfügen kann.

Die Ausgabe des Ergebnisbaumes ist wie bereits erwähnt standardmäßig wiederum eine XML-Datei. Um die Restriktion des XML-Syntax zu lockern, besteht die Möglichkeit, dem XSL-Prozessor mit `<xsl:output>` mitzuteilen, welche Art von Ausgabedatei man zu erzeugen beabsichtigt.

Das Attribut „method“ spezifiziert dabei den Ausgabebetyp. Neben dem Wert „text“ für einfache Textdateien besteht auch die Möglichkeit, mit „html“ den gelockerten HTML-Syntax zu verwenden (der z.B. mit `
` Ausnahmen vom XML-Syntax erlaubt. Die drei Modi "xml", "html" und "text" sind zwar Quasi-Standard, jedoch ist mit passendem XSL-Processor jedes beliebige Ausgabeformat möglich.

2.4 Zwei XSLT-Prozessoren im Vergleich

2.4.1 Xalan

Das Open-Source-Projekt Xalan ist der Welt des Web-Servers Apache entsprungen und bietet schon seit längerem einen stabilen, aktuellen (2001) und standard-konformen XSLT-Processor. Sein Einsatzgebiet ist vornehmlich das XSLT-Processing von Java-Servlets aus, jedoch stellt es auch ein vollwertiges Standalone-Programm nebst Java-API dar.

Das Programm selbst ist in Java geschrieben, eine C++-Variante ist in Arbeit, und benötigt mindestens JRE 1.1.8 zur Ausführung, ist also auf nahezu jeder Plattform lauffähig.

2.4.2 Start von Xalan als Standalone-Programm

Nachdem die Variable CLASSPATH richtig gesetzt ist, d.h. das Xalan-Jar-Paket enthält, sieht ein Start ungefähr so aus (in einer Befehlszeile):

```

java org.apache.xalan.xslt.Process -in source.xml
-xsl stylesheet.xsl -out target.html

```

In diesem Beispiel ist die Ausgabe eine HTML-Datei, die Art der Ausgabe hängt jedoch nicht von der Endung der Zieldatei ab, sondern wird vom XSL-Stylesheet bestimmt.

2.4.3 Start von Xalan per API-Aufruf

Das Xalan-API ist sehr umfangreich, weshalb eine Anwendung hier nur in einem sehr rudimentären Beispiel gezeigt werden kann. Das komplette API besteht aus 212 Klassen!

```
XSLTProcessor processor = XSLTProcessorFactory.getProcessor();  
processor.process(new XSLTInputSource("birds.xml"),  
                new XSLTInputSource("birds.xsl"),  
                new XSLTResultTarget("birds.out"));
```

Dies bewirkt das Gleiche wie der oben gezeigte Aufruf des Standalone-Processors, jedoch direkt aus einem Java-Programm heraus.

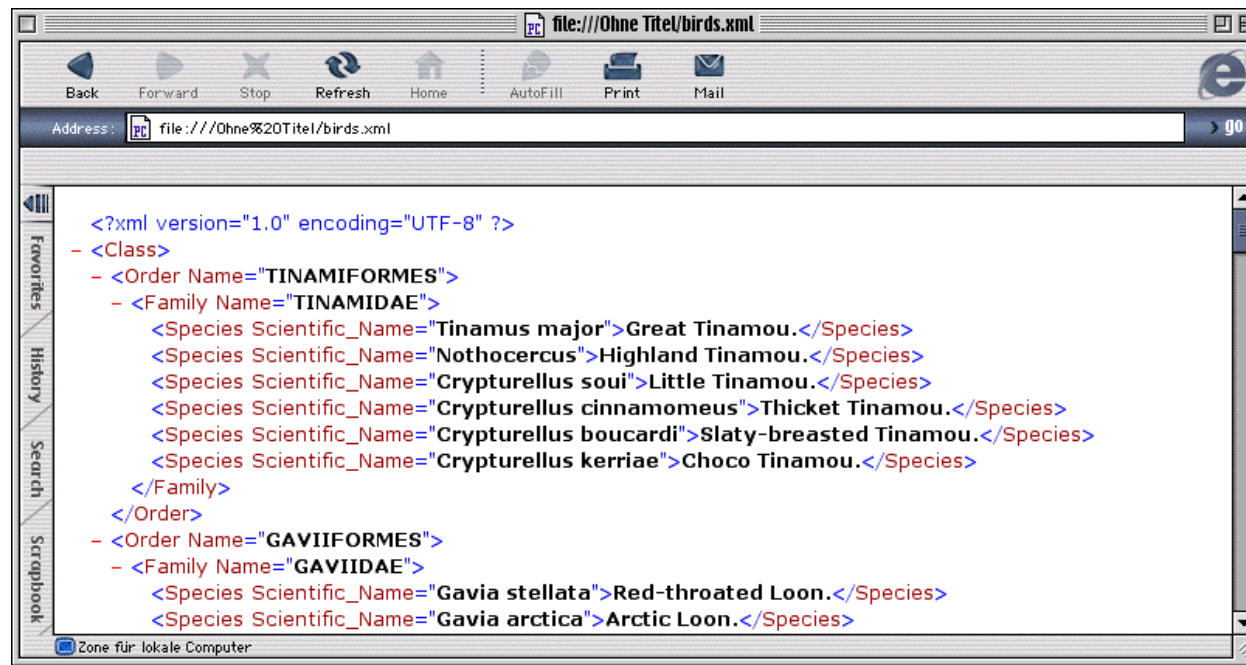
2.4.4 Internet Explorer

Trotz seiner hohen Verbreitung durch die Beigabe zu den Windows-Betriebssystemen sind die XSLT-Processor-Fähigkeiten des Internet Explorers recht unbekannt. Genau genommen liegen dessen Fähigkeiten zumindest unter Windows nicht in seinem Programmcode, sondern in Systemfunktionen vor.

Microsoft beschreitet mit diesem Produkt in mehrfacher Hinsicht einen Sonderweg:

Zum einen ist der implementierte XSLT-Standard von 1999 und nur unvollständig. Stattdessen wurde der XSLT-Processor um eigene Erweiterungen für VBScript und VisualBasic ergänzt. Zum anderen transformiert der Internet Explorer nicht in eine Datei, sondern direkt in eine Darstellung, d.h. die Kette XML & XSL in HTML, dann HTML & CSS in die Darstellung entfällt zumindest scheinbar - der Benutzer lädt nur eine XML-Datei.

Eine reine XML-Datei wird in einer Baumstruktur dargestellt, in der man Knoten beliebig auf- und zuklappen kann, um sich durch die Hierarchie zu bewegen. Die Daten erscheinen direkt, es ist keine spezielle Formatierung vorgenommen worden.

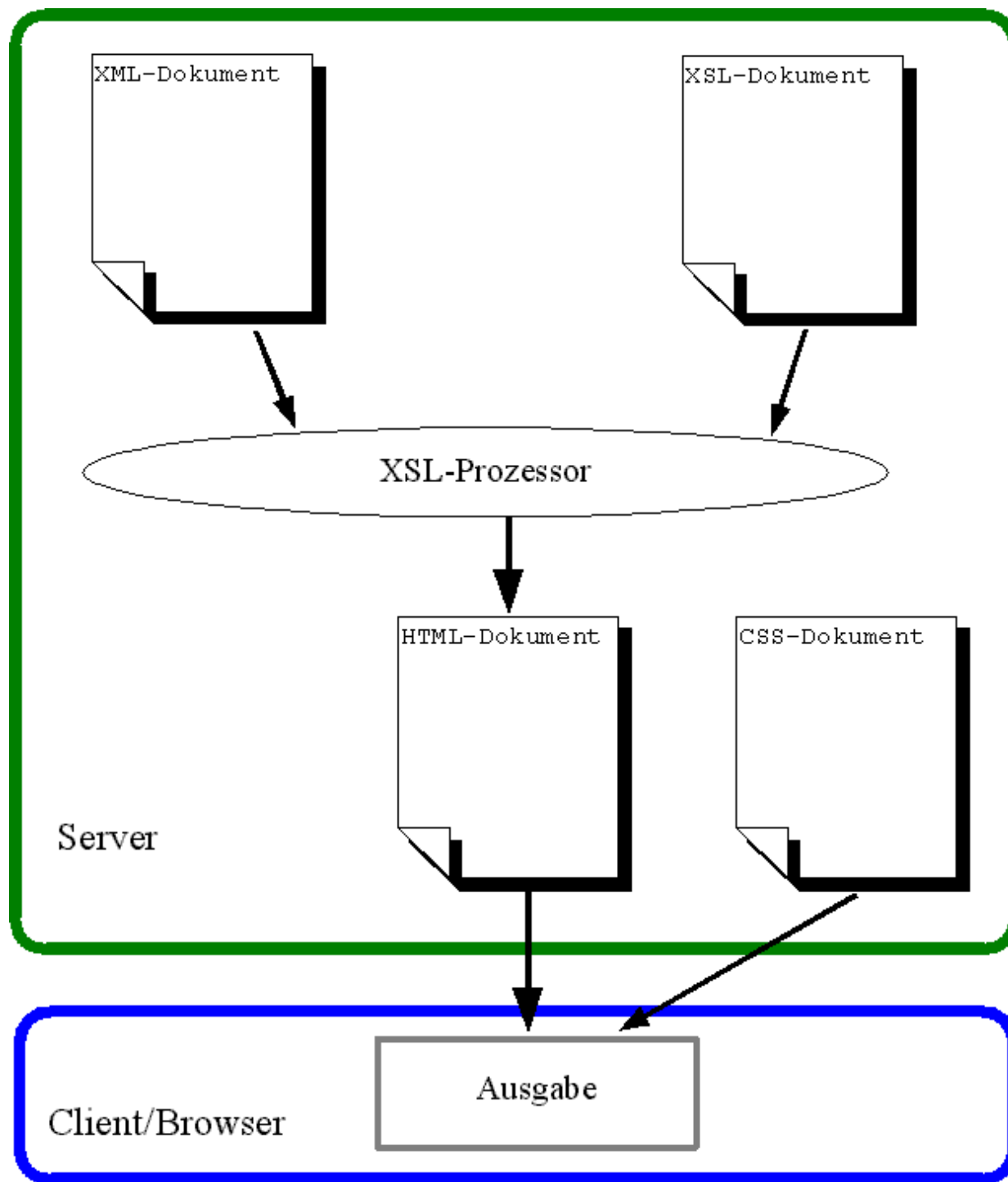


Eine XML-Datei mit einer bestimmten Processing-Instruction am Anfang, in der auf eine zu ladende XSL-Datei verwiesen wird, wird dann wie beschrieben dargestellt. Hierzu ist zu sagen, daß diese Anweisung eigentlich nicht benutzt werden sollte, da damit nahezu alle Vorteile von XML in Sachen Trennung von Daten und Darstellung ad absurdum geführt werden:

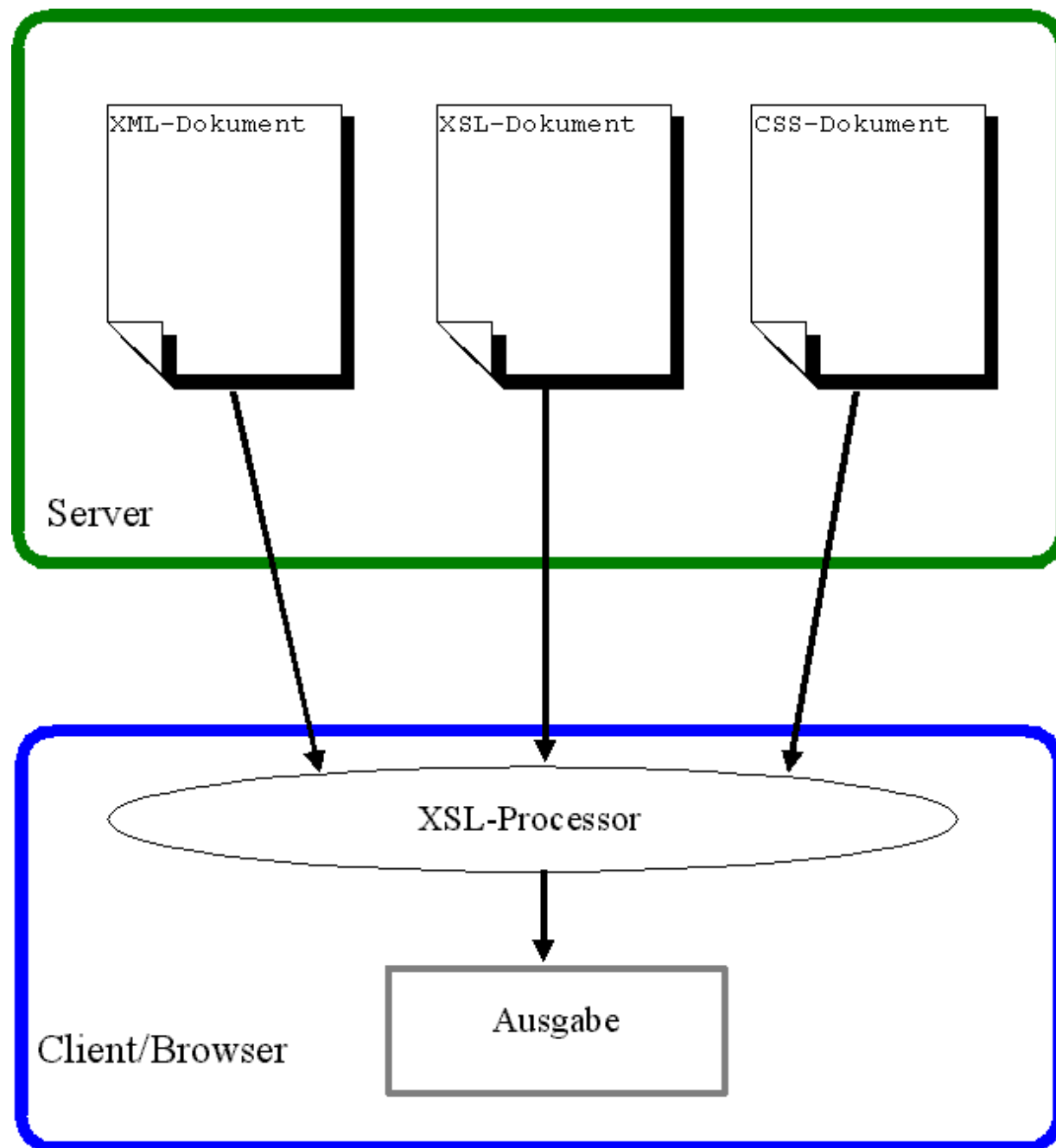
```
<?xml-stylesheet type="text/xml" href="myStyle.xsl"?>
```

Ferner nimmt man sich damit die Möglichkeit, verschiedene Sichten eines Datensatzes zu generieren.

2.4.5 Das Standardverfahren



2.4.6 Das Verfahren des Internet Explorers bei eingebetter processing instruction



2.4.7 Xalan und Internet Explorer in der Übersicht

	Xalan	Internet Explorer
Aktualität	2001	1999
XSL-T	ok	teilweise
XSL-FO	-	-
XSL-konform	ja	nein
Standalone	ja (XSL-Processor)	ja (Browser)
Serveranbindung	z.B. Apache	über VB/VBScript
Besonderheiten	in Java, mit API	

3. XSL-FO

3.1 Was ist XSL-FO?

Wie schon eingangs erwähnt, stellt XML nur die Struktur von Daten zur Verfügung, jedoch keine Art der Formatierung. Dies haben bisher stets Cascading Style Sheets übernommen, die jedoch von HTML als Quelldokument ausgehen und auch nur HTML ausgeben. HTML besitzt aber keine strikte Trennung von Daten und Darstellung.

Häufig wäre eine andere Repräsentation von XML-Daten in einem anderen Dokumenten-Format wünschenswert, z.B. PDF (Portable Document File) für den Buchsatz.

Genau hier kommt der zweite, zu Unrecht oft unbeachtete zweite Teil von XSL zum Zuge: XSL-FO (Formatted Objects). XSL-FO bietet dabei eine sowohl genaue als auch flexible Möglichkeit, XML zu repräsentieren.

XSL-FO bietet momentan 56 Elemente, mit denen man nach dem Box-Prinzip arbeitet. Ähnlich wie in TeX, PostScript oder PDF wird alles in Boxen dargestellt, die z.B. Text, Bilder oder Filme enthalten können - oder wiederum andere Boxen.

Zur Veranschaulichung: Eine Seite besteht aus mehreren Boxen, die die vier Seitenränder und den eigentlichen Seitenkörper beinhalten. Der Seitenkörper enthält selbst wiederum Boxen mit Überschrift und beliebig vielen Absatz-Boxen. Ein Bild würde als Box mit einem Bild und einer Bildunterschrift aufgefaßt.

Je nach Bedarf können die Boxen genau positioniert werden (bei HTML 4.0 immer noch ein Vabanque-Spiel), in der Regel überläßt ein XSL-FO-Programmierer jedoch die Positionierung dem XSL-FO-Transformator.

Desweiteren arbeitet XSL-FO seitenorientiert, womit XSL-FO selbst für solch anspruchsvolle Aufgaben wie den Satz eines ganzen Buches geeignet ist. Da die Seitenlänge jedoch beliebig gewählt werden kann, sind auch kleine Flugblätter oder Web-Seiten möglich, wobei es bei letzteren mit den CSS konkurrieren dürfte.

3.2 Praxis

```
<?xml version="1.0"?>
<fo:root xmlns:fo="">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="only">
      <fo:region-body/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-name="only">
    <fo:flow flow-name="xsl-region-body">
```

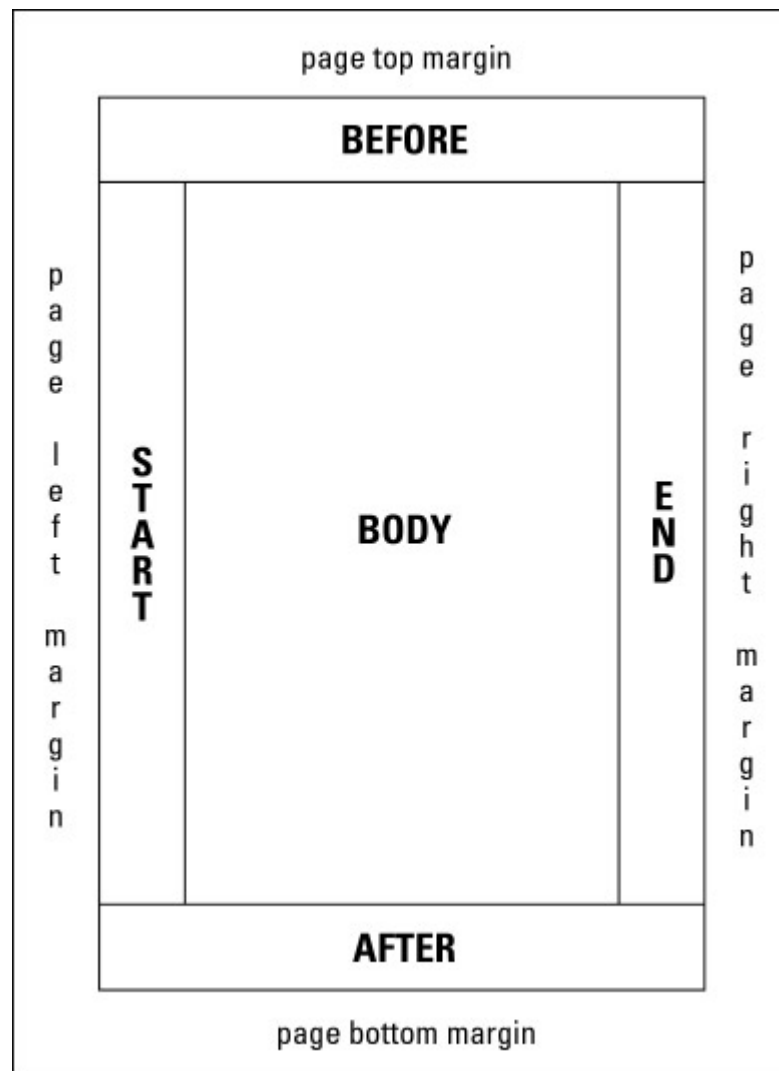
```

<fo:block font-size="20pt"
font-family="serif"
line-height="30pt">
  Germany
</fo:block>
<fo:block font-size="20pt"
font-family="serif"
line-height="30pt">
  France
</fo:block>
</fo:flow>
</fo:sequence>
</fo:root>

```

Wie immer gilt: Keine Panik! Das Element `layout-master-set` spezifiziert den Seiten-Prototyp, also das Aussehen und den Aufbau der Seiten des Dokuments. Der XSL-FO-Processor liest die Daten, die in `page-sequence` stehen, füllt eine Seite, liest weiter usf., bis keine Daten mehr zu verarbeiten sind.

Eines der Standard-Seitenlayouts sieht so aus:



Der große Nachteil im obigen Listing besteht darin, daß Daten und Darstellung nicht voneinander getrennt sind. Doch kann man ja verschiedene XML-Dokumente und -Vokabulare durch die Trennung der Namespaces mixen:

```
?xml version="1.0"?>
```

```

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">

  <xsl:template match="/">
    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
      <fo:layout-master-set>
        <fo:simple-page-master master-name="only">
          <fo:region-body/>
        </fo:simple-page-master>
      </fo:layout-master-set>
      <fo:page-sequence master-name="only">

        <fo:flow flow-name="xsl-region-body">
          <xsl:apply-templates select="//COUNTRY"/>
        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>

  <xsl:template match="COUNTRY">
    <fo:block font-size="20pt"
      font-family="serif"
      line-height="30pt">
      <xsl:value-of select="@NAME"/>
    </fo:block>
  </xsl:template>

</xsl:stylesheet>

```

Diese Vorgehensweise setzt allerdings eine von zwei Möglichkeiten voraus:

1. Zuerst wird das Dokument von einem XSL-T-Processor transformiert, danach von einem XSL-FO-Processor, oder
2. Der Processor beherrscht beide Vokabulare und das Dokument in einem Durchgang transformieren.

Leider ist momentan die erste Variante noch die gängige, da noch kein vollwertiger XSL-Processor existiert. Was gemeinhin unter XSL-Processor bekannt ist, stellt nur den XSL-T-Part dar, nicht die Formatted-Objects! Die wenigen vorhandenen XSL-FO-Processoren wandeln nur in ein Zwischenformat wie PDF um, sie stellen selbst keine Anzeige zur Verfügung.

3.3 Vergleich XSL-FO vs. CSS

	XSL-FO	CSS
Quelldokument	XML	HTML
Ausgabeformat	beliebig	HTML
Anwendungsbereich	Web bis Buchsatz	Web
Datenunabhängigkeit	ja	(ja)
Verbreitung	-	+

3.4 Fazit

Obwohl XSL-FO die wesentlich mächtigere Alternative zu CSS/HTML ist, wird man noch auf derlei Browser warten müssen. CSS-Programmierer müssten zwar umlernen, jedoch steht mit den Formatted-Objects ein Transformations- und Formattierungs-Tool

zur Verfügung, das aufgrund seiner Anwendungsbreite von simplen Flugblättern und Übersichten über Webseiten bis hin zu Buchsatz, große Zukunft hat. Damit entfällt die bisherige die bisherige Notwendigkeit, für verschiedene Ausgabemedien verschiedene Umwandlungsverfahren anzuwenden. Die Trennung zwischen Daten und Darstellung ist klar, ein grundsätzliches Layout, wie eine Corporate Identity, kann leichter ausgetauscht werden.

Anhang A: Quellen

- Seminararbeit von Dimitrio Malheiro zum Seminar: Information Processing on the Web
http://www.informatik.uni-freiburg.de/~malheiro/seminararbeit_dimi.ps.gz
- XML Bible, Second Edition
<http://www.ibiblio.org/xml/books/bible2/chapters/ch17.html>
<http://www.ibiblio.org/xml/books/bible2/chapters/ch18.html>
- Learning XSL
<http://www.w3.org/Style/XSL/#learning>
- Jeremie Tech XSL Page
<http://www.jeremie.com/JumpStart/xsl.jer>
- XSL Seite von oasis
<http://www.oasis-open.org/cover/xsl.html>
- Die XSLT-Seite
<http://www.xslt.com>