XSL - Extensible Stylesheet Language

XML-Daten transformieren und präsentieren

Michael Dienert

10. November 2014

Inhaltsverzeichnis

1 Was ist XSL

1.1 XML Stylesheet Language

- XSL ist die Abkürzung von *Extensible Stylesheet Language*.
- XSL ist selbst wieder eine XML-Anwendung, d.h. XSL-Dokumente müssen immer wohlgeformte XML-Dokumente sein.
- XSL besteht aus drei Teilen:

$$\text{XSL} = \text{XPath} + \text{XSLT} + \text{XSL-FO}$$

1.1.1 Die drei Bestandteile von XSL

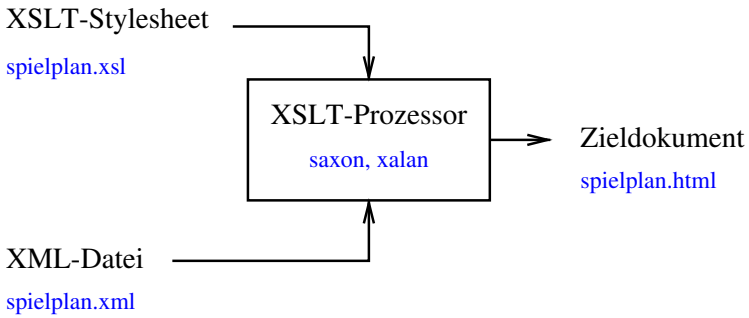
XPath: um Teile eines XML-Dokuments auswählen zu könnnen

XSLT: **XSL-T**ransformations, um ein XML-Dokument in ein anderes (XML)-Dokument transformieren zu könnnen

Hauptanwendung: Transformation von xml (Daten) nach html (Darstellung)

XSL-FO: **XSL-F**ormatting **O**bjects, um ein XML-Dokument in eine Präsentationsform wie z.B. pdf oder PostScript umwandeln zu können.

1.1.2 Der Transformationsvorgang



- Aufruf von saxon:

```
java -jar saxon.jar spielplan.xml spielplan.xsl
```

- Aufruf von xalan (C++ - Version):

```
xalan -in europe.xml -xsl europe.xsl -out test.dxf
```

1.1.3 Zusammenspiel von XPath, XSLT und XSL-FO

XPath: wird hauptsächlich **innerhalb** von XSLT-Dokumenten eingesetzt.

XSL-FO: XSL-FO-Dokumente werden von einem XSL-FO-Formatierer in z.b. pdf umgewandelt. Beispiel für ein solches Programm:

Apache Formatting Objects Processor **fop**

XSLT: XSL-FO-Dokumente werden nicht von Hand geschrieben, sondern mit XSLT aus einem XML-Dokument erzeugt.

konsequente Trennung von Daten (xml) und Darstellung (xsl)

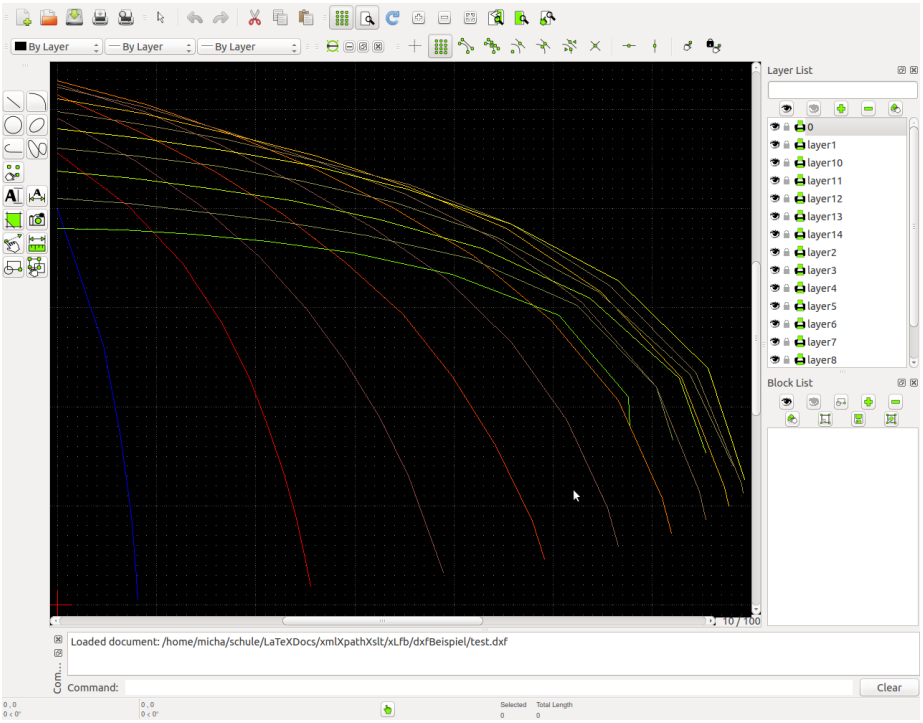
- XSLT erzeugt **beliebige** Zieldokumente
- Zum Beispiel: *Data Interchange Format*: quelloffenes CAD-Austauschformat der Firma *Autodesk*
- Gegeben sind eine Reihe von x-y-Koordinaten (Spantenriss)
- Daten liegen bereits in einem xml-Format vor
- Aufgabe: die Daten in ein CAD-Format bringen
- Der Einfachheit halber Version 12 gewählt: 1988 !

Rohdaten im xml-Format

```
<spant nr="1">
   <vertex>
      <x>81.00000000</x>
      <y>5.50000000</y>
   </vertex>
   <vertex>
      <x>80.26183187</x>
      <y>14.15232648</y>
   </vertex>
   <vertex>
      <x>79.40402646</x>
      <y>28.90070211</y>
   </vertex>
    ...
</spant>
```

Zeichnungsdaten im dxf-Format

```
    ...
0
POLYLINE
8
0
70
4
0
VERTEX
10
81.00000000
20
5.50000000
0
VERTEX
10
80.26183187
20
14.15232648
    ...
```



2 Das Spielplan-Beispiel

Im folgenden Beispiel werden die Daten eines Theaterspielplans in einer XML-Datei (`spielplan.xml`) gespeichert.

2.1 Datenhaltung in einer XML-Datei

`spielplan.xml`

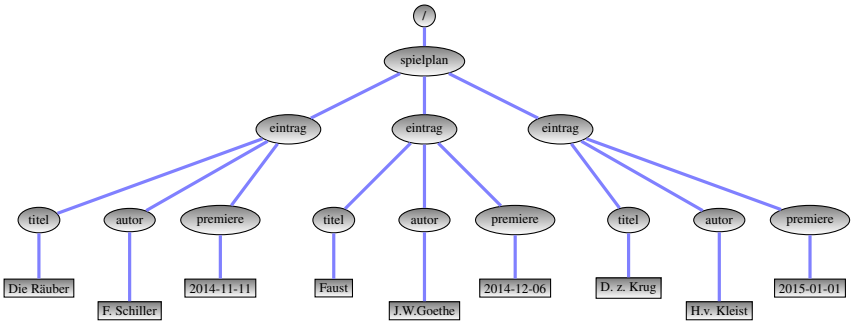
```
<?xml version="1.0" encoding="iso-8859-1"?>

<spielplan>
   <eintrag>
      <titel>Die Raueber</titel>
      <autor>Friedrich Schiller</autor>
      <premiere>2014-11-11</premiere>
   </eintrag>
   <eintrag>
      <titel>Faust</titel>
      <autor>Johann Wolfgang Goethe</autor>
      <premiere>2014-12-06</premiere>
   </eintrag>
      <eintrag>
      <titel>Der zerbrochene Krug</titel>
      <autor>Heinrich von Kleist</autor>
```

```
      <premiere>2015-01-01</premiere>
   </eintrag>
</spielplan>
```

2.2 Datenhaltung in einer XML-Datei

Aufgabe:

- Die xsl-Datei soll den xml-Spielplan so transformieren, dass der Plan als **html-Tabelle** ausgegeben wird.
- Dabei muss man sich immer wieder vergegenwärtigen, dass eine xml-Datei eine **Baumstruktur** besitzt.
- schrittweise soll ein Stylesheet, besser wäre der Name **Transformationssheet**, für die Transformation entwickelt werden.



3 Bestandteile einer XSL-Datei

3.1 Template Rules

Der wichtigste Bestandteil einer xslt-Datei sind die sog. **Template Rules**. Eine Template Rule hat folgende Syntax:

```
<xsl:template match="xpath-Lokalisierungsschritt">
...
</xsl:template>
```

Der Namensraumpräfix muss nicht **xsl** heissen, dies ist jedoch allgemeine Praxis und sinnvoll.

Die URI für diesen Namensraum ist: `http://www.w3.org/1999/XSL/Transform`

Der xpath-Lokalisierungsschritt wählt dabei den Teil des Originaldokuments aus, für den die Template Rule gelten soll. Innerhalb der Template Rule gelten nun folgende, einfache, **ganz wichtige** Regeln:

1. Führe alle Anweisungen aus, deren Markup zum xsl-Namensraum gehört
2. Gib den Rest ins Ergebnisdokument aus.

Nun möchte man evtl. Daten aus dem Quelldokument ins Zieldokument übernehmen. Das macht man mit folgendem Ausdruck:

```
<xsl:value-of select="xpath-Ausdruck"/>
```

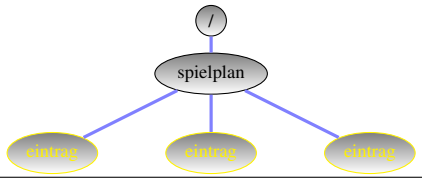
Eine Template-Rule lässt sich mit foldenden Ausdrücken aufrufen:

```
<xsl:apply-templates/>
```

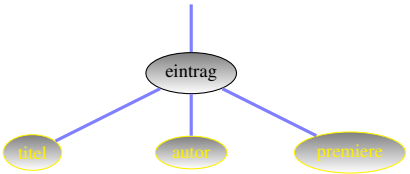
Die Wirkung dieser Anweisung ist einfach:

- selektiere **alle Kinder** des aktuellen Knotens; man erhält eine ⇒ Knotenmenge (Node-Set)
- suche für jedes Kind eine passende (gemäss match-Attribut) *Template-Rule* und führe diese aus.

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




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



Mehr Kontrolle über den Transformationsvorgang hat man mit dem optionalen Attribut select :

```
<xsl:apply-templates select="xpath-Ausdruck"/>
```

Mit dem optionalen xpath-Ausdruck lässt sich ein Kindsknoten gezielt ansteuern. Lässt man ihn weg, werden die Template Rules *aller* Kinder ausgeführt. Da das zu transformierende Dokument eine Baumstruktur besitzt (es ist ein xml-Dokument), wird bevorzugt **rekursiv** geaerbeitet.

```
<xsl:template match="eintrag">
   <tr>
      <xsl:apply-templates/>
   </tr>
</xsl:template>
```

3.2 Schrittweise Entwicklung der Spielplan-Transformationsdatei

Es soll die xml-Datei mit dem Theater-Spielplan als html-Tabelle ausgegeben werden. Zuerst ein bisschen Bürokratie:

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

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="iso-8859-1"/>

      ....

</xsl:stylesheet>
```

Da xsl-Stylesheets selbst wieder wohlgeformte xml-Dokumente sind, muss es ein Wurzelelement `<xsl:stylesheet>` geben. Das erste Template gilt für den Wurzelknoten ("/"), das gesamte Dokument (document node) selbst:

```
                    ...

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

   <xsl:template match="spielplan">
                    ...
      <xsl:apply-templates/>
                    ...
   </xsl:template>
                    ...
```

Da das Wurzel**element** keine Nachbarn hat, kann man die Templates zusammenfassen:

```
                    ...
```

```
   <xsl:template match="/spielplan">
                     ...
      <xsl:apply-templates/>
                     ...
   </xsl:template>
                     ...
```

Die nächsten Elemente im Baum und eine logische Verknüpfung:

```
                     ...

   <xsl:template match="/spielplan"> ...
   </xsl:template>

   <xsl:template match="eintrag"> ...
   </xsl:template>

   <xsl:template match="titel | autor | premiere"> ...
   </xsl:template>
                     ...
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
   <head><title>Eine erste xslt-Uebung</title></head>
   <body>
      <h1>Spielplan des Klassischen Theaters</h1>
      <table border="1">
         <tr>
            <th>Titel</th>
            <th>Autor</th>
            <th>Premiere</th>
         </tr>
         <tr>
            <td>Die Raueber</td>
            <td>Friedrich Schiller</td>
            <td>8. Juli 2010</td>
        </tr>
        <tr> ...  </tr>
        <tr> ...  </tr>
     </table>
   </body>
</html>
```

Im Template für `/spielplan`: html-head und -body deklarieren, Tabelle mit Kopfzeile einleiten, **pro Tabellenzeile nächstes Template aufrufen**

Im Template für `eintrag`: eine neue Tabellenzeile einleiten, nächstes Template aufrufen

In den Templates für `titel, autor, premiere`: eine Tabellenzelle einleiten und mit den Daten aus dem xml-Dokument füllen

Zusammenfassen gleicher Aktionen: um Redundanz zu vermeiden, können Templates mit gleichen Aktionen zusammengefasst werden

- Führe alle Anweisungen aus, deren Markup zum xsl-Namensraum gehört
- Gib den Rest ins Ergebnisdokument aus.
- Daten aus dem Quelldokument ins Zieldokument übernehmen:

```
<xsl:value-of select="xpath-Ausdruck"/>
```

```
<xsl:template match="/spielplan">
   <html>
      <head><title>Eine erste xslt-Uebung</title></head>
      <body>
         <h1>Spielplan des Klassischen Theaters</h1>
         <table border="1">
            <tr>
               <th>Titel</th>
               <th>Autor</th>
               <th>Premiere</th>
            </tr>

            <xsl:apply-templates/>

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

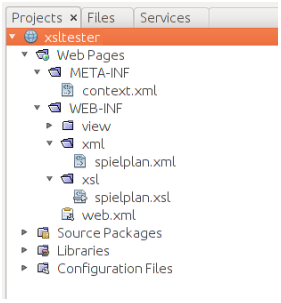
```
   <xsl:template match="eintrag">
      <tr>
         <xsl:apply-templates/>
      </tr>
   </xsl:template>

   <xsl:template match="titel | autor | premiere">
      <td><xsl:value-of select="./text()"/></td>
   </xsl:template>
```

4 Übungen zu xslt

4.1 Der Theater-Spielplan als html-Tabelle

- Um eine xsl-Datei zu testen, muss man sie zusammen mit der zugehörigen xml-Datei mit einem xsl-Prozessor wie *saxon* oder *xalan* aufrufen. Der Ausgabestrom des xsl-Prozessors kann dann wiederum z.B. mit einem html-Browser angezeigt werden.
- Um diesen Vorgang zu automatisieren und das Transformationsergebnis direkt in einem Browser anzeigen zu können, wurde eine kleine Java-Webanwendung erstellt.

- Benötigte Programme: Netbeans IDE mit Tomcat (Java EE - Download), html-Browser
- Anleitung:
- Laden Sie foldende ZIP-Datei herunter: `http://dt.wara.de/xml/xsltester.zip`
- Starten Sie `NetBeans` und importieren Sie die heruntergeladene Datei (File-Menue → Import Project → From ZIP)
- Starten Sie die Anwendung
- Erstellen Sie das gewünschte Stylesheet in der noch unvollständigen Datei `spielplan.xsl` (zum Editieren mit Doppelklick öffnen):



4.2 Eine kurze Erklärung zu Java-Web-Applikationen

- Java-Web-Applikationen verwenden einen in Java geschriebenen Server, der direkt Java-Code ausführen kann und der das http-Protokoll implementiert.
- Bekannteste Java-Webserver:
  - Tomcat (apache): unterstützt Servlets und JSP-Standards, Vorteil: leichtgewichtig
  - Glassfish (Sun/Oracle): fast komplette Unterstützung des Java-EE-Standards, Nachteil: sehr resourcenhungrig
- Als xsl-Prozessor werden Klassen aus der Java-Bibliothek `javax.xml.transform` verwendet. Hiermit ist eine xsl-Transformation direkt in der Web-Applikation möglich. Das Ergebnis wird im Browser angezeigt.

4.3 Übungen

- Erstellen Sie ein xsl-Stylesheet, das die Theaterspielplan-xml-Datei in eine html-Tabelle überführt.
- Je nach Arbeitstempo kann die Tabelle zunehmend aufwändig gestaltet und sortiert werden:

XSLT macht Spass!

Titel	Autor	Premiere
Die Räuber	Friedrich Schiller	2014-10-11
Faust	Johann Wolfgang Goethe	2014-11-12
Der zerbrochene Krug	Heinrich von Kleist	2014-12-13
Tod eines Handlungsreisenden	Henry Miller	2015-02-02
Der Watzmann	Ambros/Tauchen/Prokopetz	2014-11-22
Der Vorname	Delaporte/Patellière	2014-12-13

XSLT macht Spass!

Nr.	Titel	Autor	Premiere
1	Die Räuber	Friedrich Schiller	2014-10-11
2	Faust	Johann Wolfgang Goethe	2014-11-12
3	Der zerbrochene Krug	Heinrich von Kleist	2014-12-13
4	Tod eines Handlungsreisenden	Henry Miller	2015-02-02
5	Der Watzmann	Ambros/Tauchen/Prokopetz	2014-11-22
6	Der Vorname	Delaporte/Patellière	2014-12-13

- Und schliesslich mit Sortierung:

XSLT macht Spass!

Nr.	Titel	Autor	Premiere
1	Die Räuber	Friedrich Schiller	2014-10-11
2	Faust	Johann Wolfgang Goethe	2014-11-12
3	Der Watzmann	Ambros/Tauchen/Prokopetz	2014-11-22
4	Der Vorname	Delaporte/Patellière	2014-12-13
5	Der zerbrochene Krug	Heinrich von Kleist	2014-12-13
6	Tod eines Handlungsreisenden	Henry Miller	2015-02-02

- die fortlaufende Nummer wird mit der Funktion position() erzeugt. position() enthält die Nummer des gegenwärtigen Knotens.
- für die Hintergrundfarben der Zeilen benötigen Sie:

xsl:variable : Erzeugen Sie eine Variable mit Namen `"farbe"` und weisen Sie dieser die Hintergrundfarbe zu.

```
http://www.w3schools.com/xsl/el_variable.asp
```

css-style : Das Attribut `style` wird verwendet, um die Hintergrundfarbe einer html-Tabellenzeile zu setzen:

```
<tr style="background-color:{$farbe};">
```

xsl:choose : Unterschiedliche Werte für die Farbvariable wählen (gerade/ungerade Zeilennummern);

```
http://www.w3schools.com/xsl/xsl_choose.asp
```

Zu Vergleichs- und Modulo-Operatoren siehe hier:

```
http://www.w3schools.com/xpath/xpath_operators.asp
```

xsl:sort : das <xsl:sort>-Element darf nur als *erstes* Kindelement innerhalb des <xsl:apply-templates>- oder des <xsl:for-each>-Elements (siehe unten) stehen. Genaueres hier:

```
http://www.w3schools.com/xpath/xpath_operators.asp
```

xsl:if : Der Vollständigkeit halber:

```
http://www.w3schools.com/xsl/xsl_if.asp
```

5 Noch mehr xsl-Elemente

5.1 XSLT-Schleifen und Pull-Processing

- XSLT-Schleifen:

```
<xsl: for-each select="xpath-Ausdruck"
```

- der select-Ausdruck wählt eine Knotenmenge aus
- über die Knotenmenge wird mit for-each iteriert
- ⇒ `<xsl:for-each>` bildet eine Alternative zu `<xsl:apply-templates>`
- man *zieht* die ausgewählten Knoten in ein Template hinein
- man nennt diesen Ansatz: *Pull Processing*

```
<xsl:template match="/spielplan">
   ...
   <xsl:for-each select="eintrag">
      <tr><td>
            <xsl:value-of select="titel"/>
          <td></td>
            <xsl:value-of select="autor"/>
          <td></td>
            <xsl:value-of select="premiere"/>
      </td></tr>
   </xsl:for-each>
   ...
</xsl:template>
```

⇒ Die Knoten `titel`, `autor` und `premiere` werden ins Template von `/spielplan` *hineingezogen*.

5.2 Erzeugen von Attributen und Elementen

- Problem: wir möchten html-Elemente mit Attributen erzeugen. Die Werte der Attribute sollen aus der xml-Datei stammen.
- Beispiel-Daten (xml): die Spielplan-Datei wird um `<szenenfotos>`-Elemente erweitert:

```
       ...
<eintrag>
   <titel>Die Räuber</titel>
   <autor>Friedrich Schiller</autor>
   <premiere>2014-10-11</premiere>
   <szenenfotos>raueber.jpg</szenenfotos>
</eintrag>
       ...
```

1. **Attribute Value Templates**

```
      <a href="{xpath-Ausdruck}"> Linkname </a>
```

   Die geschweifen Klammern weisen den XSLT-Prozessor an, den enthaltenen xpath-Ausdruck auszuwerten und das Ergebnis dort in den Ausgabestrom zu schreiben.

2. `<xsl:attribute>`

```
      <xsl:attribute name="href"><xsl:value-of
      select="xpath-ausdruck"/> | text></xsl:attribute>
```

   - Vorsicht: keine Zeilenumbrüche in die Attributwerte einbauen!
   - Vorteil: kann mit Kontrollstrukturen verwendet werden
   - Siehe auch:

     ```
     http://www.w3schools.com/xsl/el_attribute.asp
     ```

1. Erzeugen Sie Links in der html-Tabelle mit `Attribute Value Templates`
2. Erzeugen Sie die Links nur, wenn die xml-Datei die nötigen Dateipfade enthält. Geben Sie andernfalls einen Hinweis aus (mit <xsl:attribute> und <xsl:choose>):

XSLT macht Spass!

Nr.	Titel	Autor	Premiere	Szenenfotos
1	Die Räuber	Friedrich Schiller	2014-10-11	klick: Szenefotos
2	Faust	Johann Wolfgang Goethe	2014-11-12	keine aktuellen Bilder vorhanden
3	Der Watzmann	Ambros/Tauchen/Prokopetz	2014-11-22	keine aktuellen Bilder vorhanden
4	Der Vorname	Delaporte/Patellière	2014-12-13	klick: Szenefotos
5	Der zerbrochene Krug	Heinrich von Kleist	2014-12-13	keine aktuellen Bilder vorhanden
6	Tod eines Handlungsreisenden	Henry Miller	2015-02-02	keine aktuellen Bilder vorhanden

- Im Ausgabedokument sollen (xml)-Elemente vorkommen, deren Namen erst zur Laufzeit des xsl-Transformationsvorgangs festgelegt werden.
- Erzeugen mit `<xsl:element>` und ggfs `<xsl:attribute-set>` :

```
<xsl:attribute-set name="attrSetName">
   <xsl:attribute name="name">
      attribut-wert
   </xsl:attribute>
</xsl:attribute>

<xsl:element name="elementname"
             use-attribute-sets="attrSetName1, ...">
   ...
</xsl:element>
```

- Anwendung 1: Erzeugen von xsl-Dateien mit xsl-Transformationen
- Anwendung 2: Verwendung von Attribute-Sets
- Übung (optional): die Tabellenzeilen abwechselnd einfärben. Dabei `<xsl:attribute-set>` für die Farben und `<xsl:element>` für die Table-Row (tr) verwenden.

6 Programmierung mit XSLT

- Template-Rules können das Attribut *name* erhalten und können dann über den Namen aufgerufen werden. ⇒ Template-Rules entsprechen somit Funktionen, Methoden.
- Template-Rules können auch Parameter erhalten
- Entsprechend kann eine Template-Rule mit Parametern aufgerufen werden
- Syntax und Verwendung folgen in einem Beispiel
- Definition einer Variablen, Variante1:

```
<xsl:variable name="name" select="ausdruck" />
```

- Definition, Variante2:

```
<xsl:variable name="name">
    anweisungen
</xsl:variable>
```

- Geltungsbereich: innerhalb des umschliessenden Elements, aber dort nur nach ihrer Definition
- Unglaublich aber wahr: Variablen können nach ihrer Definition nicht mehr verändert werden: Immutable Variables

- Unveränderbare “Variablen” sind eine grundlegende Eigenschaft von rein **funktionalen** Sprachen.
- Problem: mit xslt soll der Inhalt eines Webshop-Warenkorbs tabellarisch dargestellt werden.
- In der letzten Tabellenzeile soll der Gesamtwarenwert angezeigt werden.
- Aufsummieren in einer nicht veränderbaren Variablen ist nicht möglich!
- Lösung: rekursive Programmierung.
- Pseudocode:

```
function total(liste) {
  if (liste != leer) {
    summe = total(liste ab element 2);
    return summe +
      (1. elem. der liste).preis *
      (1. elem. der liste).bestellmenge;
  }
  else
    return 0;
}
```

```
<xsl:template name="total">
 <xsl:param name="liste" />
  <xsl:choose>
   <xsl:when test="$liste">
    <xsl:variable name="summe">
     <xsl:call-template name="total">
      <xsl:with-param name="liste" select="$liste[position() &gt;
           1]" />
       </xsl:call-template>
    </xsl:variable>
    <xsl:value-of select="$summe + $liste/preis * $liste/
        bestellmenge" />
   </xsl:when>
  <xsl:otherwise>0</xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

7 Ein ganz kurzer Ausflug zu XSL-FO

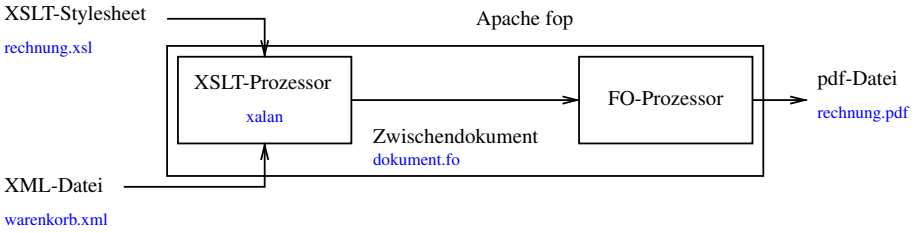
7.1 Ausgabe von Druckerzeugnissen

- xml is slightly verbose but highly structured
- xsl-fo is **ultra** verbose
- xsl-fo basiert auf xml: xsl-fo-Dateien sind xml-Dateien
- xsl-fo ist weit besser als html geeignet, einen text milimetergenau für die Druckausgabe zu formatieren. D.h.:

  - html: Bildschirmausgabe
  - xsl-fo: Ausgabe von Druckerzeugnissen

- dabei ist xsl-fo weniger eine Seitenbeschreibungssprache, sondern wurde für grosse, komplexe Dokumente konzipiert.
- z.Zt. verstehen Drucker xsl-fo nicht direkt, daher wird xsl-fo nach pdf gewandelt (Formatting Objects Prozessor fop)

7.1.1 Der Formatting Objects Transformationsvorgang



Aufruf:

```
fop -xml warenkorb.xml -xsl rechnung.xsl -pdf rechnung.pdf
```

7.1.2 Ergebnis der Transformation


libro it

alfred neumann, waelderstrasse 7, 79341 kenzingen

Zeile1
Zeile2
Zeile3
Zeile4
Zeile5
Zeile6

sachberarbeiter
telefon
emaille-adr
datum

bezugszeichenzeile

Sehr geehrte Damen und Herren,

Pos.	Art.-Nr.	Autor	Titel	Aufl.	Verlag	Menge	Einzel-preis	Preis in EUR
1	978-3-540-73338-6	P. Gantenoco, W. Alex	Debian GNU/ Linux	3	Springer	50	59.95	2997.50
2	978-0-470-19274-0	Michael Kay	XSLT 2.0 and XPATH 2.0	4	Wiley Publishing	1	49.90	49.90
3	3-8266-7209-7	F.J. Herpers, T.J. Sebesteyen	XSL, Das Einsteigerseminar	1	verlag moderne industrie	1	9.95	9.95
4	978-3-540-24524-7	Benedikt Stockebrand	IPv6 in Practice	1	Springer	1	9.95	9.95
Gesamtsumme:								3067.30

bankverbindungen:
sparkasse freiburg - konto 987654321 - blz 68050101
postbank karlsruhe - konto 123456789 - blz 66010075

8 Java Architecture for XML-Binding

8.1 Übersicht zu JAXB

- Von XML zu Java: Generierung von Java-Code aus einem XSD-Schema
- Und zurück: Generierung eines XSD-Schemas aus Java-Code (machen wir nicht)
- Serialisierung eines Java-Objekts in ein XML-Dokument: Marshalling
- Erzeugung eines Java-Objekts aus XML-Daten: Unmarshalling

XSD-Datei (gekürzt):

```
<xs:schema ... >
<xs:element name="kontakte">
 <xs:complexType><xs:sequence>
  <xs:element name="Person" maxOccurs="unbounded">
   <xs:complexType><xs:sequence>
     <xs:element name="name" type="xs:string" />
     <xs:element name="adresse" type="xs:string" minOccurs="0" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:long" use="required" />
   </xs:complexType> ...
```

Beispiel einer nach diesem XSD-Schema *validen* XML-Datei:

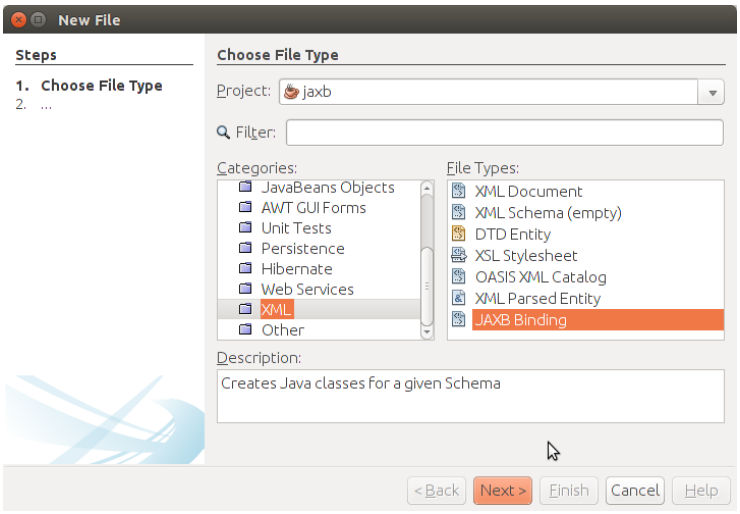
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<kontakte>
    <Person id="19631223">
        <name>dienert, michael</name>
        <adresse>waelderstrasse 7, 79341 kenzingen</adresse>
    </Person>
    <Person id="19700101">
        <name>neuman, alfred e.</name>
        <adresse>locaRoca 666, 1234 dancelingen</adresse>
    </Person>
    <Person id="20000210">
        <name>bosak, jon</name>
        <adresse>
           4150 Network Cir, Santa Clara, CA USA
        </adresse>
    </Person>
</kontakte>
```

Java-Code generieren mit dem Kommando `xjc`:

```
xjc kontakte.xsd
```

Direktes Erzeugen der Quellen innerhalb eines NB-Projekts:

File-Menue → New File → Categorie: XML → Type: JAXB Binding



- Die XSD-Datei beschreibt eine xml-Datei, die beliebig viele (`maxOccurs="unbounded"`) <Person>-Elemente enthalten darf.
- Jedes <Person>-Element muss ein Atrribut **id** haben.
- Jedes <Person>-Element enthält **genau ein** (weder minOccurs noch maxOccurs vorhanden) Element <name>.
- Jedes <Person>-Element enthält **optional** (`minOccurs=0`, maxOccurs nicht vorhanden, d.h `maxOccurs="1"`) ein Element <adresse>.
- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelelements: `Kontakte.java`
- Die vielen <Personen>-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`
- Marshalling: ähnlich Serialisierung von Objekten; Orignalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
// create JAXB context and instantiate marshaller
JAXBContext context = JAXBContext.newInstance(Kontakte.class);
Marshaller m = context.createMarshaller();
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);

// Write to System.out
m.marshal(kontakte, System.out);

// Write to File
m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier:`Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
private List<Kontakte.Person> personListe;

 JAXBContext context = JAXBContext.newInstance(Kontakte.class);
 Unmarshaller u = context.createUnmarshaller();

 Kontakte kontakteNeu = (Kontakte) u.unmarshal(
    new File("/home/micha/andereKontakte.xml"));

  personListe = kontakteNeu.getPerson();
```