



Transformation von XML-Dokumenten

Was ist XSL – politisch?

- XSL ist eine eigene Technik.
- Zum Publizieren (Transformieren) von Dokumenten.
- Früher gab es dafür Verlage und Schriftsetzer, um gute Inhalte in gute Produkte zu verwandeln.
- Seit Word müssen wir das alles selber machen.
- Das führt in die falsche Richtung, XML/XSLT/XQuery verschärft das Problem noch.
- Jeder, der mit XML gute Inhalte erstellt sollte einen neben sich sitzen haben, der daraus gute Produkte erzielt.

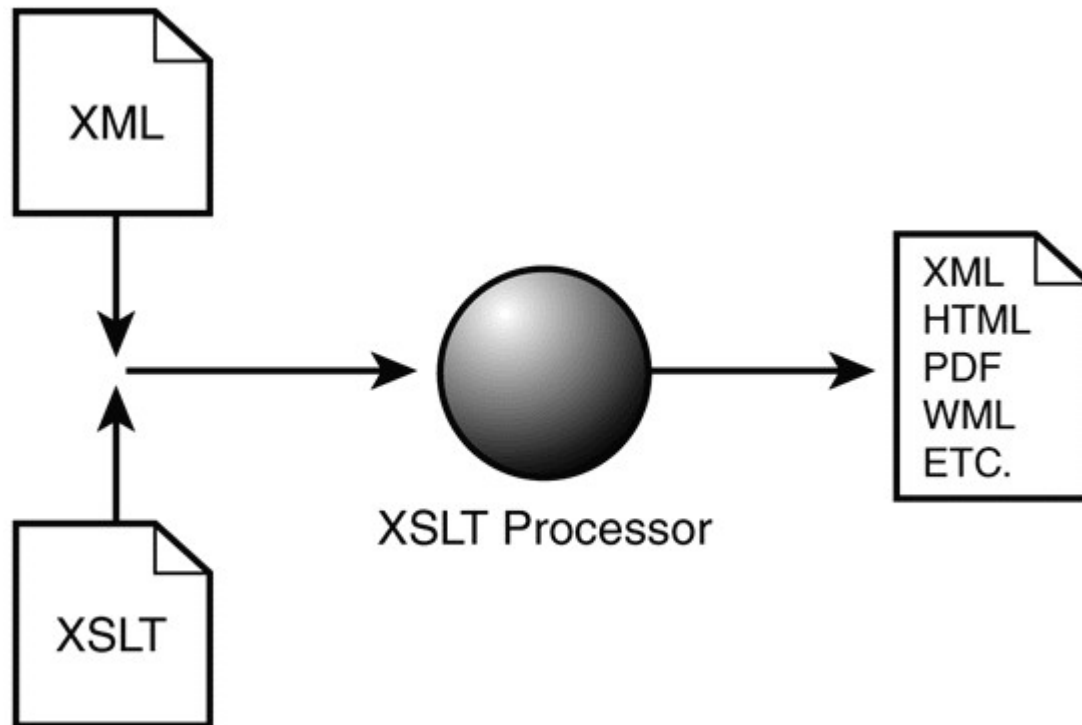
Was ist XSL – technisch?

- Mehrere Komponenten:
 - XSLT → Transformations
 - XSL-FO → Formatting Objects

- Ausgabeformate von XSLT:
 - (x)HTML
 - XML → z.B. Word, TEI, RDF, SVG
 - Text → z.B. LaTeX (→ PDF), RTF, MARC

- Ausgabeformat von XSL-FO:
 - XML → PostScript → PDF

XSLT-Verarbeitung



XML: Vor- und Nachteile

- Vorteil: Die Trennung von Struktur, Inhalt und Aussehen.
- Nachteil: Die Trennung von Struktur, Inhalt und Aussehen.

- Problem: Wenn ein Ergebnis ausgegeben werden soll, muss man die Zielsprache auch kennen!
 - HTML, CSS, (JavaScript)

- Problem: Alles, was ein Ausgabedokument „leisten“ soll, muss in der Kodierung vorhanden sein.
 - z.B. Register, Verknüpfungen, Formatierungen

Strukturen im XML

- Abschnitte, Absätze (Unterscheidung physikalische, inhaltliche Strukturen, phrase-level)
- Listen (Aufzählung, Bibliographie, etc), Tabellen
- Bilder
- textuelle Eingriffe (Ersetzung, Abkürzung)
- Seitenumbrüche
- Fußnoten, Referenzen
- Register
- Sonderzeichen / Unicode
- weitere?



Strukturen im HTML

- Abschnitte, Absätze, Überschriften
- Listen (Aufzählung, Bibliographie, etc), Tabellen
- Bilder
- Sonderzeichen / Unicode
- Formatierungen (CSS)
- weitere?



Ein Transformationsergebnis: HTML

- Die wichtigsten HTML-Elemente
 - html, head, body
 - div, p, h1-6
 - span, pre
 - ul, ol, li
 - table, tr, td

- CSS (Cascading Stylesheets)
 - font-weight:bold
 - font-size:smaller
 - font-style:italic

- <http://de.selfhtml.org>

XSL (eXtensible Stylesheet Language)

- Der Prozessor traversiert den Baum!
 - Immer, wenn ein neues Ereignis wahrgenommen wird (neues Element, Attribut, Text), wird in der XSL-Datei nachgeschaut, ob eine Regel angewendet werden muss.

- Woher weiß der Prozessor, wann er was machen soll?
 - **XPath**
 - `<xsl:template match="" />`

XPath

- Technologie zur Adressierung von Knoten
- Enthält nützliche Funktionen
- Achsen-basierte Navigation durch die Hierarchieebenen

Ohne XPath ist die Abfrage und Verarbeitung von XML-Dokumenten unmöglich!

XPath Knotentypen

7 Knotentypen

- Dokument- und Elementknoten `<person>...</person>`
- Text- und Attributknoten `<el att=„value“>text</el>`
- Namensraumknoten `<tei:lb/>`
- Processing-Instruction-Knoten `<? ... ?>`
- Kommentarknoten `<!-- ... -->`



XPath Notation

- Nach dem Prinzip: **achse::knotentest([prädikat])**
 - /
 - /child::persName
 - /descendant::persName
 - /parent::*
 - /child::person/attribute::role
 - /self::TEI/child::*

XPath verkürzte Notation

- Nach dem Prinzip: **achse::knotentest([prädikat])**
 - / → /
 - /child::persName → /persName
 - /descendant::persName → //persName
 - /parent::* → ..
 - /child::person/attribute::role → /person/@role
 - /self::TEI/child::* → ./*



XPath Prädikate

- Bedingungen, die Submengen definieren
 - `//person[@role = "member"]`
 - `/persName[surname = "müller"]`
 - `/persName[surname != "müller"]`
 - `//person[1]`
 - `//person[last()]/persName`
 - `//person[position() > 5]`

XPath Funktionen

- Dienen zur weiteren Verarbeitung und Adressierung von Knotenmengen

- 4 Gruppen von Funktionen
 - Knotenmengenfunktionen**
 - Zeichenkettenfunktionen**
 - Logische (boolsche) Funktionen
 - Numerische Funktionen

XPath Funktionen

- Knotenmengen
 - last(), position(), current(), count(), ...
- Zeichenketten
 - concat()
 - contains(), matches()
 - starts-with(), ends-with()
 - substring(), substring-before(), substring-after()
 - translate()

xsl:stylesheet

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

... Anweisungen hierher ...

```
</xsl:stylesheet>
```

- Wichtig: TEI-Dateien müssen mit TEI-Namespace verarbeitet werden!

```
<xsl:stylesheet version="1.0"  
  xmlns:tei="http://www.tei-c.org/ns/1.0"  
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
</xsl:stylesheet>
```

XSLT Beispiel

- XML: `<message>Hallo Welt!</message>`
- XSLT:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html><body>
      <h1><xsl:value-of select="message"/></h1>
    </body></html>
  </xsl:template>
</xsl:stylesheet>
```

- Der HTML-Namespace wird als default angenommen!

Das Grundgerüst: Templates

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

- Schablone als Anweisung, an welcher Stelle im XML-Dokument (match="XPath") etwas gemacht werden soll

Die „push“-Methode

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

- Anweisung, dass die Verarbeitung weiterer Templates fortgeführt werden soll.
- Optional kann die zu verarbeitende **Knotenmenge** über das **@select** per XPath bestimmt werden.

Beispiel:

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

Die „pull“-Methode

```
<xsl:call-templates name=" name-eines-templates "/>
```

- Templates können gezielt aufgerufen werden.
- Anwendung: Registerbildung

Beispiel:

```
<xsl:template match="/">  
  <xsl:call-template name=" irgendein-name ">  
</xsl:template>  
<xsl:template name="irgendein-name ">  
  ... macht was ...  
</xsl:template>
```



Ausgabe von Text

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

- Wählt den **Wert** eines XPath-Ausdrucks, also **Text** eines Elements oder Attributs und fügt diesen in die Ausgabe ein.
- `<h1> <xsl:value-of select="message"/> </h1>`

`<xsl:text>` Schreibe diesen Text ... `</xsl:text>`

Schleifen

```
<xsl:for-each select=" xpath-ausdruck ">  
  ...  
</xsl:for-each>
```

- Beispiel: Wähle jeden Text (`//msItem`) und generiere eine Aufzählung (``) der Titel (`title`):

```
<ul>  
  <xsl:for-each select="//msItem">  
    <li> <xsl:value-of select="title"/> </li>  
  </xsl:for-each>  
</ul>
```




Bedingungen I

```
<xsl:if test=" xpath-ausdruck "> ... </xsl:if>
```

- Führt Anweisungen nur aus, wenn die Bedingung in @test erfüllt ist.

Beispiel:

```
<xsl:for-each select="//book">  
  <xsl:if test=" author = 'Cassiodor' ">  
    <li><xsl:value-of select="title"/></li>  
  </xsl:if>  
</xsl:for-each>
```

Bedingungen II

```
<xsl:choose>
```

```
  <xsl:when test=" xpath-ausdruck " > ... </xsl:when>
```

```
  <xsl:otherwise></xsl:otherwise>
```

```
</xsl:choose>
```

- Unterscheidet mehrere Fälle, in denen Anweisungen nur ausgeführt werden, wenn die Bedingung in @test erfüllt ist. Ansonsten werden die Anweisungen des xsl:otherwise ausgeführt.

Sortieren

`<xsl:sort select=" xpath-ausdruck "/>`

- Sortiert die Elemente einer Schleife (nach dem ausgewählten Kriterium)
- Wird immer innerhalb von `<xsl:for-each>` oder `<xsl:apply-templates>` verwendet

Beispiel:

```
<ul>  
  <xsl:for-each select="//msItem">  
    <xsl:sort select="author"/>  
    <li> <xsl:value-of select="title"/>  
      <xsl:text> von </xsl:text>  
      <xsl:value-of select="author"/> </li>  
  </xsl:for-each>  
</ul>
```

Kopieren

`<xsl:copy/>`

- Kopiert den aktuellen Knoten inklusive Namespace
- Kopiert keine Attribute, Inhalte, Kindknoten

`<xsl:copy-of select="xpath-ausdruck"/>`

- Kopiert den ausgewählten Knoten inklusive aller Kindknoten und Attribute

Variablen / Parameter

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

- wird einmalig festgelegt

```
<xsl:param name="" />
```

- kann „unterwegs“ verändert werden