Texttransformation mit XSLT

Patrick Sahle
sahle@uni-koeln.de

Ulrike Henny
ulrike.henny@uni-koeln.de

Texttransformation mit XSLT

- Was ist XSLT? Wozu braucht man XSLT? Wie funktioniert XSLT?
- Ein Beispiel
- Übung: Hallo Welt
- XSLT in oXygen
- Die wichtigsten Elemente
- Übung 1: Ortsliste
- Übung 2: Ausgabe Postkartentext

Was ist XSLT?

- eXtensible Stylesheet Language Transformation
- W3C Standard seit 1999
- Sprache zur Transformation von XML-Dokumenten
- XSL ist auch XML
- Transformationsszenario:
  - XML-Dokument → XSLT-Stylesheet → Zieldokument(e)
  - Eingabebaum → Transformationsbaum → Ausgabebaum/-dokument
  - Verschiedene Zielformate: XML, (X)HTML, Text, …

Wozu braucht man XSLT?

- Verwalten von Dokumenten
  - Aufräumen von Daten
  - Anreicherung mit neuen Daten
  - Zusammenführen mehrerer Dokumente
  - Aufspalten in einzelne Dokumente
- Umwandlung für den Datenaustausch
- Aufbereitung zur Darstellung und Publikation
  - Auswahl treffen
  - Organisieren
  - Schön machen
  - Funktional machen
  - Ergänzen

Wie funktioniert XSLT?

- Transformation von Dokumenten mit XSLT = Umformung ihrer Struktur
- Schablonen / Vorlagen / Templates
  - XPath zum Zugriff auf den Eingabebaum
  - Elemente, Attribute und Text, der ausgegeben werden soll
- Weitere XSLT-Elemente, mit denen die Datenauswahl und –ausgabe gesteuert wird

Ein Beispiel - XSLT

- Ich bin ein XML-Dokument

  Ich bin ein XSL-Stylesheet

  Ich bin eine Schablone. Ich passe auf ein Muster.

  Ich tue etwas: Ich hole etwas aus dem Ausgangsdokument, ich schreibe etwas in das Zieldokument.

Ein Beispiel - XSLT

- Ich bin ein XML-Dokument

Ich bin ein XSL-Stylesheet:

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">

Ich bin eine Schablone. Ich passe auf ein Muster:

<xsl:template match="XPath-Ausdruck">

Ich tue etwas: Ich hole etwas aus dem Ausgangsdokument, ich schreibe etwas in das Zieldokument: *TEXT*

</xsl:template>

</xsl:stylesheet>

Übung: Hallo Welt

- Ziel:
  - „Hallo Welt!“ mit XSLT
- Vorgehen:
  - oXygen öffnen, XSLT-Datei erstellen
  - Template erstellen
  - Text in das Zieldokument schreiben
  - XSLT-Datei mit dem XML-Ausgangsdokument verknüpfen
  - Transformation ausführen

XSLT in oXygen

- Neue XSLT-Datei anlegen:
  - Menü: Datei → Neue Datei → XSLT-Stylesheet → Erstellen


- XML-Dokument mit XSLT-Stylesheet verknüpfen:
  - XML-Dokument öffnen
  - Menü: Dokument → XML-Dokument → XSLT/CSS-Stylesheet zuordnen

- Transformation ausführen:
  - XML-Dokument mit Transformationsanweisung öffnen
  - Dokument → Transformation → Transformation-Szenarios anwenden

XSLT in oXygen

- Transformationsszenario anpassen:
  - Dokument → Transformation → Transformation-Szenarios konfigurieren 
- Zieldokument formatieren:
  - Dokument → Quelle → Dokument formatieren 

Die wichtigsten Elemente: Templates

- <xsl:template match="XPath-Muster" name="String" mode="String">
  - Eine Schablone, die alles verarbeitet, was auf das XPath-Muster passt. Schablonen können benannt sein oder einen bestimmten Modus haben.
  - Mindestens @match oder @name muss vorhanden sein.
- <xsl:apply-templates select="XPath-Ausdruck" mode="String" />
  - Aufruf in einer Schablone, der besagt, dass für diese Eingabedaten noch weitere Schablonen berücksichtigt werden sollen.
  - Optional mit @select: für welche Elemente genau sollen weitere Schablonen berücksichtigt werden?
  - Optional mit @mode: nur Schablonen mit diesem Modus sollen berücksichtigt werden
- <xsl:call-template name="String" />
  - Aufruf einer anderen Schablone mit ihrem Namen

Templates: Beispiel

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

<xsl:template match="dokumentinhalt">
    <xsl:apply-templates select="zeile[not(@typ='vordruck')]" />
</xsl:template>

<xsl:template match="vorname[@geschlecht='männlich']">
      Dieser männliche Vorname kommt auf der Postkarte vor: …
</xsl:template>
```

Die wichtigsten Elemente: Wertausgabe & Kopieren

- <xsl:value-of select="XPath-Ausdruck" />
  - Schreibt den Wert (den Text) von bestimmten Teilen des Eingabedokuments (@select) in das Ergebnisdokument
  - Kurzsyntax innerhalb von Attributkonstruktionen: attribut= "{XPath-Ausdruck} "
- <xsl:copy-of select="XPath-Ausdruck" />
  - Kopiert die mit @select ausgewählten Elemente vollständig (mit allen Attributen und Kindelementen)
- <xsl:copy />
  - Kopiert nur das aktuelle Element (ohne Attribute und ohne Kindelemente)

Wertausgabe & Kopieren: Beispiele

```
<xsl:template match="name">
    <xsl:value-of select="nachname" />, <xsl:value-of select="vorname" />
</xsl:template

<xsl:template match=“adresse“>
    <empfänger postkarte=“{ancestor::Postkarte//titel}“>
        <xsl:copy-of select="//anrede" />
        <xsl:copy-of select=“//empfänger/name“ />
    </empfänger>
</xsl:template>
```

Die wichtigsten Elemente: Elemente & Attribute

- <xsl:element name="String">
  - Konstruiert ein Element im Ergebnisbaum
  - Elemente können auch direkt in das Zieldokument geschrieben werden: <elementname>elementinhalt</elementname>
- <xsl:attribute name="String" />
  - Konstruiert ein Attribut im Ergebnisbaum
  - Attribute können auch direkt in das Zieldokument geschrieben werden: <elementname attributname="attributwert">elementinhalt</elementname>
  - Wertausgabe im Attribut: <elementname attributname="{XPath-Ausdruck}">
- <xsl:text>
  - Schreibt Zeichendaten in das Zieldokument

Die wichtigsten Elemente: Schleifen & Sortieren

- <xsl:for-each select="XPath-Ausdruck" >
  - Mit jedem Element, das vom XPath-Ausdruck zurückgegeben wird, soll etwas getan werden

- <xsl:sort select= "XPath-Ausdruck" data-type= "text | number" order="ascending | descending" case-order="upper-first | lower-first" lang="de | en | …" />
  - Sortiert die Knotenmenge, die mit @select ausgewählt wird
  - Kann vorkommen in: <xsl:for-each>, <xsl:apply-templates>
  - Wird als leeres Element direkt nach dem öffenden Tag von z.b. for-each notiert

Schleifen & Sortieren: Beispiel

```
<xsl:template match="Postkarte">

    <xsl:for-each select="name">
        <xsl:sort select="nachname" />
        <xsl:value-of select="nachname" />, <xsl:value-of select="vorname" />
    </xsl:for-each>

</xsl:template>
```

Die wichtigsten Elemente: Bedingungen

- <xsl:if test="XPath-Ausdruck">
  - Bedingung: die in <xsl:if> enthaltenen Anweisungen werden nur ausgeführt, wenn der Ausdruck in @test wahr ist

- <xsl:choose>
- <xsl:when test="XPath-Ausdruck"></xsl:when>
- <xsl:otherwise></xsl:otherwise>
- </xsl:choose>
  - Es werden mehrere Bedingungen nacheinander (<xsl:when test=" ">) überprüft und die Anweisungen der ersten erfüllten Bedingung ausgeführt.
  - Trifft keine der Bedingungen in @test zu, werden die Anweisungen in <xsl:otherwise> ausgeführt.

Bedingungen: Beispiel

```
<xsl:template match="Postkarte">
    <xsl:if test="contains(.//vorderseite, 'Dortmund')">
        Dies ist eine Dortmund-Postkarte.
    </xsl:if>
</xsl:template>
```

Die wichtigsten Elemente: Zieldokumente

- <xsl:result-document href="String">
  - Erstellen eines Zieldokuments
  - In @href können der Pfad und Dateiname des Zieldokuments angegeben werden
  - Dadurch Erstellen mehrere Zieldokumente möglich

- <xsl:output method="html | text | xhtml | xml" encoding=" UTF-8 | ISO-8859-1" indent="yes | no">
  - Gibt an, wie der Ergebnisbaum/das Zieldokument ausgegeben werden soll
  - Top-Level-Element!
  - @method: Angabe des Formats
  - @encoding gibt die Zeichencodierung an
  - @indent: bei „yes“ werden die Elemente im Ergebnisbaum eingerückt (für lesbaren Quelltext)

Zieldokumente: Beispiel

- <xsl:template match="Postkarte">
  - <xsl:result-document href="vorderseite.xml">
  - Hier wird ein eigenes Dokument für die Vorderseite
  - der Postkarte erstellt.
  - </xsl:result-document>
  - <xsl:result-document href="rückseite.xml">
  - Hier wird ein eigenes Dokument für die Rückseite
  - der Postkarte erstellt.
  - </xsl:result-document>
- </xsl:template>

Die wichtigsten Elemente: Leerraum

- <xsl:strip-space elements="String"/>
  - Leerraum, der im Ausgangsdokument zwischen Elementen stand, wird entfernt
  - In @elements wird eine Liste von Elementnamen angegeben, getrennt durch Leerzeichen
  - Top-Level-Element
- <xsl:preserve-space elements="String"/>
  - Leerraum, der im Ausgangsdokument zwischen Elementen stand, wird beibehalten
  - In @elements wird eine Liste von Elementnamen angegeben, getrennt durch Leerzeichen
  - Top-Level-Element

Leerraum: Beispiel

- <name geschlecht="weiblich">
  <vorname>Margarete</vorname>
  <nachname>Grogorenz</nachname>
  </name>

- <xsl:strip-space elements="vorname nachname“/>
- → MargareteGrogorenz
- <xsl:preserve-space elements="vorname nachname"/>

→ Margarete
Grogorenz

Übung 1: Ortsliste

- Ziel:
  - Postkarte von [Verfasser] an [Adressat]
    Orte: [Ort1], [Ort2], …, [Ort x]

Übung 1: Ortsliste

Transformationsanweisung

Schablone, trifft auf Dokument zu
Postkarte von [Absender] an [Empfänger]

Orte: Für jeden Ort (jeder Ort nur einmal; und sortiere die Orte) …
gib den Ort aus; schreib ein Komma dahinter (wenn es nicht der letzte Ort ist)

Übung 1: Ortsliste

- Vorgehen?
- Für die Überschrift
  - Baue ein Template, das
  - …auf das Dokument „passt“
  - …den Text „Postkarte von … an … “ schreibt
  - …nach „von“ den Wert des Elementes <verfasser> ausgibt
  - …nach „an“ den Wert des Elementes <adressat> ausgibt
- Und in XSLT?
  - <xsl:template>
  - <xsl:value-of select="XPath-Ausdruck" />
- XPath:
  - .//

Übung 1: Ortsliste

- Für die Liste der Orte
  - Schreibe den Text „Orte: “ in das Template
  - Nimm von allen <ort>-Elementen diejenigen, die zum ersten Mal vorkommen [XPath: wenn es kein vorheriges <ort>-Element gibt, das den gleichen Wert hat wie das aktuelle]
  - Sortiere die Orte
  - Schreibe jeden Ort in das Zieldokument
  - Wenn es nicht der letzte Ort in der Liste ist, dann füge ein Komma und ein Leerzeichen hinzu

Übung 1: Ortsliste

- Und in XSLT?
  - <xsl:for-each select="XPath-Ausdruck">
  - <xsl:sort />
  - <xsl:value-of select="´XPath-Ausdruck" />
  - <xsl:if test="XPath-Ausdruck">
- XPath:
  - .//
  - […]
  - not()
  - Preceding::elementname
  - =, !=
  - .
  - position()
  - last()

Übung 1: Lösung

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="2.0">
        <xsl:template match="/" >
          Postkarte von <xsl:value-of select=".//verfasser"/> an
         <xsl:value-of  select=".//adressat" />

      Orte: <xsl:for-each select=".//ort[not(preceding::ort = .)]">
                <xsl:sort />
                <xsl:value-of select="." /><xsl:if test="position() != last()">, </xsl:if>
              </xsl:for-each>
        </xsl:template>
</xsl:stylesheet>
```

Übung 2: Ausgabe Postkartentext

- Ziel:
  - 1 Mein liebes Gretchen!
    2 Soeben bin ich "glücklich [ironisch]" in Dortmund
    3 angekommen und denke gleich an Dich!!
    4 Die Fahrt war "ekelhaft schneidig."
    5 Hier in Dortmund habe ich 2 Stunden
    6 Aufenthalt. Bei meiner Ankunft in
    7 Wetter schreibe ich Dir gleich. Alles
    8 andere brieflich. Sei jetzt recht herzlich
    9 gegrüßt von Deinem Walter.
    10 Tausend Grüße an deine lieben
    11 Eltern u.Paul

Übung 2: Ausgabe Postkartentext

- Vorgehen?
  - Schreibe ein Template für die Wurzel, das
    - weitere Templates aufruft
    - aber nur für den Haupttext!
  - Schreibe ein Template für die Zeilen, das
    - die aktuelle Zeilennummer ausgibt [XPath: wie viele <zeile>-Elemente gab es vorher? Plus 1]
    - weitere Templates aufruft
  - Schreibe ein Template für <ironisch>, das
    - weitere Templates aufruft
    - danach den Text [ironisch] ins Zieldokument schreibt
  - Schreibe ein Template für <durchgestrichen>, das nichts tut

Übung 2: Ausgabe Postkartentext

- Und in XSLT?
  - <xsl:template match="XPath-Ausdruck">
  - <xsl:apply-templates select="XPath-Ausdruck" />
  - <xsl:value-of select="XPath-Ausdruck" />
  - <xsl:text>
- XPath
  - //
  - count()
  - preceding-sibling::elementname
  - +

Übung 2: Lösung

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
    <xsl:template match="/">
        <xsl:apply-templates select="//haupttext" />
    </xsl:template>

    <xsl:template match="zeile">
        <xsl:value-of select="count(preceding-sibling::zeile) + 1"/><xsl:text>
</xsl:text><xsl:apply-templates />
    </xsl:template>

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

    <xsl:template match="durchgestrichen"></xsl:template>
</xsl:stylesheet>
```

Referenzen / Literatur

- Kurzes W3C Tutorial: http://www.w3schools.com/xsl/default.asp
- XSLT bei SelfHTML (auf deutsch): http://de.selfhtml.org/xml/darstellung
- W3C-Empfehlung für XSLT 1.0: http://www.w3.org/TR/xslt
- W3C-Empfehlung für XSLT 2.0: http://www.w3.org/TR/xslt20
- Bongers, Frank, XSLT 2.0 & XPath 2.0, Galileo Press 2008².
- Kay, Michael, *XSLT 2.0 and XPath 2.0 Programmer's Reference (Programmer to Programmer)*, Wiley Publishing 2008⁴.
- Koch, Daniel, *XSLT. Schnell + kompakt*, entwickler.press 2007.