Eine Schnelleinführung in XSL

Michael Dienert

8. Juli 2014

1 Was ist XSL?

- 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 den drei Teilen:
  1. **XPath**, um Teile eines XML-Dokuments auswählen zu könnnen
  2. XSL-Transformations, **XSLT**, um ein XML-Dokument in ein anderes (XML)-Dokument transformieren zu könnnen
  3. XSL-Formatting Objects, **XSL-FO**, um ein XML-Dokument in eine Präsentationsform wie z.B. pdf oder PostScript umwandeln zu können.

XPath wird hauptsächlich innerhalb von XSLT-Dokumenten eingesetzt. Und mit XSLT-Dokumenten lassen sich wiederum XSL-FO-Dokumente erzeugen, die von einem XSL-FO-Formatierer in z.b. pdf gesetzt werden können.
Im folgenden Text wird es hauptsächlich um XSLT gehen. Dabei wird XPath als bekannt vorausgesetzt.

2 XSLT-Prozessoren

Die Abb. 1 zeigt den Ablauf einer Transformation.
Der XSLT-Prozessor ist ein Programm, das den eigentlichen Transformationsvorgang ausführt. Als typischer Vertreter seien hier die XSLT-Prozessoren **saxon** und **xalan** genannt.
Xalan ist ein Projekt der *Apache Software Foundation* und wurde ursprünglich in Java geschrieben. Inzwischen gibt es den XSLT-Prozessor auch in einer C++-Version. Das Beispiel aus der Abbildung lässt sich auf der Kommandozeile so ausführen:

```
java -jar xalan.jar -IN spielplan.xml -XSL spielplan.xsl
```

Das Transformationsergebnis landet in diesem Fall in der Standardausgabe. Mit dem Parameter -OUT kann man einen Zielpfad angeben.

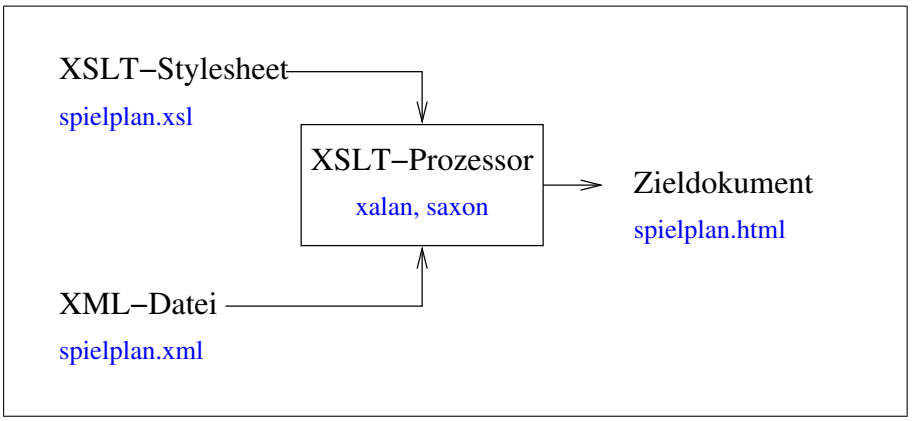

Abbildung 1: Der XSL-Transformationsvorgang

3 Das Spielplan-Beispiel

Mit einem Beispiel soll nun gezeigt werden, wie XSLT-Stylesheets programmiert werden.

Dazu soll der Spielplan eines Theaters streng getrennt nach Datenhaltung (XML-Datei) und Darstellung (XSL-Stylesheet) verwendet werden.

Die Datei spielplan.xml mit den Daten könnte z.B. so aussehen:

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

<spielplan>
   <eintrag>
      <titel>Die Räuber</titel>
      <autor>Friedrich Schiller</autor>
      <premiere>8.2.2007</premiere>
   </eintrag>
   <eintrag>
      <titel>Faust</titel>
      <autor>Johann Wolfgang Goethe</autor>
      <premiere>20.7.2007</premiere>
   </eintrag>
      <eintrag>
      <titel>Der zerbrochene Krug</titel>
      <autor>Heinrich von Kleist</autor>
      <premiere>23.11.2007</premiere>
   </eintrag>
</spielplan>
```

Die XSL-Datei soll den XML-Spielplan nun so transformieren, dass der Plan als HTML-Tabelle ausgegeben wird.

> Dabei muss man sich immer wieder vergegenwärtigen, dass eine xml-Datei eine **Baumstruktur** besitzt.

3.1 Die Hauptbestandteile einer xslt-Datei

3.1.1 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 Regeln:

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

Damit ergibt sich die allgemeine Struktur einer XSLT-Datei:

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

   <xsl:output method="html" encoding="iso-8859-1"/>

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

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

   ....

</xsl:stylesheet>
```

3.1.2 Werte aus der Quelle ins Ziel übernehmen

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

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

Dabei werden nur die Werte von XML-Elementen, also die *Textknoten* in den Ausgabestrom geschrieben.

3.1.3 Erstes Beispiel

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

Wirkung dieser Template-Regel:

- suche im XML-Dokument den Markup mit den Tags `<titel>` oder `<autor>`
- schreibe den Text `<td style="text-align:center">` ins Zieldokument
- übernehme den Wert des Textknotens, der Kind es aktuellen Elements (`<titel>` oder `<autor>`) ist ins Zieldokument
- schreibe `</td>` ins Zieldokument

Ausschnitt aus dem Zieldokument:

```
<td style="text-align:center">Die R&auml;uber</td>
<td style="text-align:center">Friedrich Schiller</td>
```

Der Umlaut in *Räuber* wird vom XSLT-Prozessor automatisch in entsprechende HTML-Entität umgesetzt, wenn im Stylesheet folgende Zeile enthalten ist:

```
<xsl:output method="html" encoding="iso-8859-1"/>
```

3.1.4 Rekursives Abarbeiten des Quelldokuments

Um das Quelldokument abzuabeiten, wird nun bevorzugt *rekursiv* [1]gearbeitet, indem man folgende Anweisung **innerhalb** von Template-Regeln plaziert:

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

Die Wirkung dieser Anweisung ist einfach: suche nach Template-Regeln für die **Kinder** des aktuellen Knotens und führe diese aus.

> Bei der Erstellung von XSL-Stylesheets muss man sich also immer wieder vergegenwärtigen, an welchem Knoten man im Baum des Quelldokuments gerade ist.

Mit einem optionalen *xpath-Ausdruck* lässt sich ein Kindsknoten auch gezielt ansteuern. Lässt man ihn weg, werden die Template Rules *aller* Kinder ausgeführt.

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

[1] ein iteratives Arbeiten ist mit dem Element `<xsl:for-each/>` ebenfalls möglich.

4 Direkte Ausführung der Transformation in einem Web-Server

Wer das im Anhang enthaltene Beispiel von saxon oder xalan transformieren lässt, erhält eine (x)html Datei. Um zu prüfen, wie diese in einem Browser aussieht, muss das Ergebnis der saxon-Transformation als html-Datei gespeichert und anschliessend mit einem Web-Broser geöffnet werden.

Ideal wäre es nun, wenn das Transformationsergebnis nicht als Datei in die Standard-ausgabe geschrieben, sondern schön ins HTTP-Protokoll verpackt als tcp-Strom nach Port 80 geschickt wird.

4.1 Eine einfache JSTL-Anwendung mit NetBeans

Mit NetBeans lässt sich auf einfache Weise eine Web-Anwendung erstellen, mit der innerhalb einer Web-Seite direkte eine XSL-Transformation angzeigt werden kann.

Da das Erstellen von Web-Anwendungen mit NetBeans hier nicht besprochen werden soll, gibt es hier eine fertige Anwendung

```
http://www.dienert.eu/~dienert/pdf/sae/xml/xslTesterNB.zip
```

Die zip-Datei muss heruntergeladen in nach Netbeans importiert werden. In den beiden Verzeichnissen

```
WEB-INF/xml
WEB-INF/xsl
```

können dann das XML-Dokument und das XSL-Stylesheet abgelegt werden.

Nach einer Änderung am XML-Dokument oder am Stylesheet genügt es, die Datei zu speichern und die Seite im Web-Browser neu zu laden. Es ist nicht notwendig, aus Netbeans heraus das Projekt neu zu starten.

5 Aufaben

Um mit XSLT vertraut zu werden, sollen unten stehende Aufgaben gelöst werden.

Als Nachschlagewerk, welche XSLT-Elemente es gibt, eignet sich folgende Seite gut:

```
http://www.w3schools.com/xsl/xsl_w3celementref.asp
```

Eine Referenz zu XPATH-Funktionen gibt es hier:

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

1. Es soll ein XML-Dokument entworfen werden, dass Daten enthält, die in einer Tabelle dargestellt werden können. Hierbei kann man sich am Beispiel spielplan.xml orientieren, allerdings dürfen die Namen der spielplan-Elemente nicht verwendet werden.

   Die Datei kann direkt mit NetBeans in der Test-Webanwendung erstellt werden. NetBeans unterstützt XML.

2. Mit einer XSLT-Datei sollen nun die XML-Daten in einer einfachen HTML-Tabelle dargestellt werden.

3. Eine Spalte der Tabelle soll in einer anderen Textfarbe gesetzt werden

4. Die erste Spalte in der Tabelle soll eine fortlaufende Nummer erhalten, die mit der XPath-Funktion `position()` erzeugt werden soll.

5. Jede zweite Zeile der Tabelle soll durch die Hintergrundfarbe hervorgehoben werden.

   Das soll dann etwa so aussehen, wie in Abb. 2

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	Die Räuber	Friedrich Schiller	2014-10-11
5	Faust	Johann Wolfgang Goethe	2014-11-12
6	Der zerbrochene Krug	Heinrich von Kleist	2014-12-13

Abbildung 2: Der Theaterspielplan als html-Tabelle

Hinweise:

Für dies Aufgabe benötigt man eine sog. XSL-Variable:

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

Auf die Variable kann dann z.b. in einer CSS-Anweisung so zugegriffen werden:

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

Um eine bedingte Ausführung zu erreichen, gibt es das Element:

```
<xsl:choose>

  <!-- Content:(xsl:when+,xsl:otherwise?) -->

</xsl:choose>
```

A Beispieldateien

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

<spielplan>
   <eintrag>
      <titel>Die Raueber</titel>
      <autor>Friedrich Schiller</autor>
      <premiere>17.2.2007</premiere>
   </eintrag>
   <eintrag>
      <titel>Faust</titel>
      <autor>Johann Wolfgang Goethe</autor>
      <premiere>20.7.2007</premiere>
   </eintrag>
      <eintrag>
      <titel>Der zerbrochene Krug</titel>
      <autor>Heinrich von Kleist</autor>
      <premiere>23.11.2007</premiere>
   </eintrag>
</spielplan>
```

Abbildung 3: Die Datei spielplan.xml

```
<?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:template match="/child::spielplan">

      <html xmlns="http://www.w3.org/1999/xhtml">
         <head>
            <title>
               Eine erste xslt-Uebung
            </title>
         </head>
         <body>
            <h1>Spielplan des Klassischen Theaters</h1>
         </body>

         <table border="1">
            <tr>
               <td>Titel</td>
               <td>Autor</td>
               <td>Premiere</td>
            </tr>
            <xsl:apply-templates/>
         </table>
      </html>

   </xsl:template>

   <xsl:template match="child::eintrag">
      <tr>
         <td><xsl:value-of select="child::titel/child::text()"/></td>
         <td><xsl:value-of select="child::autor/child::text()"/></td>
         <td><xsl:value-of select="child::premiere/child::text()"/></td>
      </tr>
   </xsl:template>

</xsl:stylesheet>
```

Abbildung 4: Die Datei spielplan.xsl