XML-Praxis

Einführung in XSLT

Jörn Clausen
joern@TechFak.Uni-Bielefeld.DE

Übersicht

- Extensible Stylesheet Language – XSL
- XSL Transformations – XSLT
- templates
- Verarbeitung steuern

XML – Und dann?

- XML beschreibt Inhalt und Struktur
- Was ist mit der Semantik?
- Was *bedeutet* `title`?

```
<title>XML &amp;amp; Friends for Dummies</title>
```

- Und hier?

```
<person><title>Prof.</title> <name>Knuth</name></person>
```

- einfachere Frage: Wie soll `title` dargestellt werden?
- Formatierung durch *style sheet*
- Extensible Stylesheet Language (XSL)

Eigenschaften von XSL

- Ursprünge:
  - CSS (Cascading Style Sheets)
  - DSSSL (Document Style and Semantics Specification Language)
- zwei Teile:
  - XSLT (XSL Transformations)
  - XSL-FO (XSL Formatting Objects)
- Transformation XML $\rightarrow$ XML
- Spezialfall: XML $\rightarrow$ HTML

Eigenschaften von XSLT

- Transformation zwischen XML-Bäumen
- vollständige Programmiersprache
- Bezeichnung „style sheet“ hat sich trotzdem gehalten
- deklarativ, ohne Seiteneffekte
- ähnlich zu funktionalen Sprachen (Haskell)
- regelbasiert
- ist selber XML
- mehrere XSLT-Prozessoren zur Auswahl

ein einfaches Stylesheet

```
<?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="/presentation">
    <html>
      <head>
        <title>Presentation</title>
      </head>
      <body>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

HTML-Ausgabe

```
<html>
  <head>
    <meta content="text/html; charset=ISO-8859-1"
          http-equiv="Content-Type">
    <title>Presentation</title>
  </head>
  <body>
  </body>
</html>
```

Aufgaben

- Die Datei `poem2html.xsl` enthält den Rumpf einer XSLT-Datei. Sieh Dir die Datei an. Verwende den XSLT-Prozessor `xsltproc`, um sie auszuprobieren:

  ```
  $ xsltproc poem2html.xsl gedicht1a.xml
  ```

  Was passiert?

- Füge ein template ein, um die wichtigsten Elemente einer HTML-Datei zu erzeugen. Lenke die Ausgabe in eine Datei um:

  ```
  $ xsltproc poem2html.xsl gedicht1a.xml > gedicht1a.html
  ```

  und sieh Dir die HTML-Datei mit Hilfe eines Web-Browsers an.

templates

- XSLT-Prozessor verarbeitet Dokument entsprechend Baumstruktur
- zum aktuellen Knoten passendes *template* wird ausgeführt

```
<xsl:template match="/presentation">
  <html>
    <head>...</head>
    <body bgcolor="white">
      ...
    </body>
  </html>
</xsl:template>
```

- Text innerhalb des templates wird ausgegeben
- XSLT-Anweisungen werden verarbeitet

templates kombinieren

- Kontrolle an XSLT-Prozessor zurückgeben

```
<xsl:template match="/presentation">
  <html>
    <head>...</head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="title">
  <h1>...</h1>
</xsl:template>
```

- rekursive Verarbeitung des nächsten Knoten

matches

- match-Attribut „echter“ XPath-Ausdruck
- Präsentations-Titel vs. Folien-Titel:

```
<xsl:template match="presentation/title">
  <h1><xsl:apply-templates/></h1>
</xsl:template>

<xsl:template match="slide/title">
  <h2><xsl:apply-templates/></h2>
</xsl:template>
```

- Prädikate:

```
<xsl:template match="slide[title/@toc='yes']">
```

- auf Schachtelung der Quotes achten

Aufgaben

- Füge `<xsl:apply-templates/>` an der passenden Stelle in `poem2html.xsl` ein. Ergänze es um ein template für `verse`-Elemente:

```
<xsl:template match="verse">
  Strophe
  <blockquote>
  </blockquote>
</xsl:template>
```

  und rufe `xsltproc` auf.

- Füge ein `<xsl:apply-templates/>` in das `blockquote`-Element ein. Was passiert?

- Der textuelle Inhalt der `line`-Elemente wird ausgegeben. Die `line`-Tags selber werden wieder entfernt.

- Kontrolle an XSLT-Prozessor zurückgeben:

```
<xsl:template match="/poem">
  <html>
    <head>
      <title>Ein Gedicht</title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
```

default rules

- Knoten soll verarbeitet werden, aber kein template vorhanden
- *default rules*

element/root node	verarbeite Kind-Knoten
text node	gib Text aus
comment node	ignorieren

- Erinnerung: Attribute sind nicht Kinder ihrer Väter!
- Stylesheets können top-down erstellt werden
- „leeres“ XSLT-Skript entfernt alles bis auf Text
- Achtung: Verarbeitung muß tatsächlich angestoßen werden

templates kombinieren, cont.

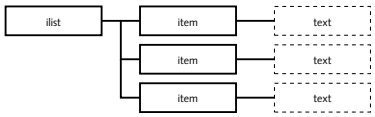
```
<ilist>
  <item>XML is ...</item>
  <item>XML instances ...</item>
</ilist>
```

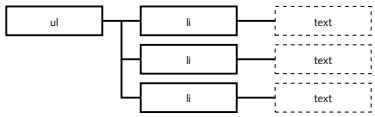
```
<ul>
  <li>XML is ...</li>
  <li>XML instances ...</li>
</ul>
```

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

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

der Transformationsprozeß






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

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

- default rule für Text-Knoten

Aufgaben

- Vervollständige `poem2html.xsl`, so daß folgender HTML-Code entsteht:

```
<h1>Der König Erl</h1>
<em>Heinz Erhardt</em>
<blockquote>
  Wer reitet so spät ...<br/>
  Es ist der Vater ...<br/>
  Im Arm den Knaben ...<br/>
  er hält ihn warm, ...
</blockquote>
```

  Beachte, daß die letzte Zeile kein `br`-Tag enthält.

- Transformiere auch die anderen Gedichte mit diesem Stylesheet.

```
<xsl:template match="title">
  <h1><xsl:apply-templates/></h1>
</xsl:template>

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

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

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

<xsl:template match="line[last()]">
  <xsl:apply-templates/>
</xsl:template>
```

Aufgaben

- Versuche, folgende Ausgabe zu erzeugen:

```
<h1>Der König Erl (<em>Heinz Erhardt</em>)</h1>
```

- Was für Probleme treten dabei auf?

Mit den bisher gezeigten Möglichkeiten ist diese Transformation nicht zu bewerkstelligen. Wenn die zu transformierende XML-Datei eine etwas andere Struktur hätte, ließe sich das Problem lösen:

```
<poem>
  <info>
    <title>Der König Erl</title>
    <author>Heinz Erhardt</author>
  </info>
  ...
</poem>
```

In diesem Fall könnte man das `h1`-Element im `info`-template erzeugen und für `title` die default-Regeln verwenden.

Es kann sich also lohnen, bereits bei der Erstellung einer XML-Datei die spätere Verarbeitung zu berücksichtigen.

XSLT ist XML

- funktioniert nicht:

```
<xsl:template match="title">
  <h1><xsl:apply-templates/>_____
</xsl:template>

<xsl:template match="author">
  (<em><xsl:apply-templates/></em>)</h1>
</xsl:template>
```

- Stylesheet muß wohlgeformt sein
- XSLT transformiert Bäume
- nur mit `<xsl:apply-templates/>` keine wirkliche Änderung der Baum-Topologie möglich

Daten auswählen

- Verarbeitung in Dokumentreihenfolge nicht immer sinnvoll
- HTML-Seite mit Titel:

```
<xsl:template match="/presentation">
  <html>
    <head>
      <title>
        <xsl:value-of select="title"/>
      </title>
    </head>
    ...
</xsl:template>
```

- verarbeiteter Knoten ist Kontext-Knoten

Daten auswählen, cont.

- Warum nicht

```
<xsl:value-of select="title/text()"/>
```

- Was passiert hier?

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

- *string-value* ist Konkatenation des enthaltenen Texts
- bzw. Attribut-Wert:

```
<xsl:value-of select="/presentation/@date"/>
```

Aufgaben

- Realisiere die Ausgabe

```
<h1>Der König Erl (<em>Heinz Erhardt</em>)</h1>
```

  mit Hilfe von `<xsl:value-of select="..."/>`.

- In welchem template sollte diese Ausgabe sinnvollerweise erzeugt werden? Wie sehen die XPath-Ausdrücke aus?
- Welches Problem tritt nun auf? Wie läßt es sich lösen?


```
<xsl:template match="title|author"/>
```

Durch „leere“ templates läßt sich die Ausgabe im zweiten Fall unterdrücken:

- Der Titel und der Autor werden zweimal ausgegeben: Einmal durch das `xsl:value-of`, andererseits wenn das Element durch `xsl:apply-templates` verarbeitet wird.

```
</xsl:template>
      (<em><xsl:value-of select="../author"/></em>)</h1>
   <h1><xsl:apply-templates/>
<xsl:template match="title">
```

- Auch diese Lösung ist möglich, aber nicht ganz so elegant:

```
</xsl:template>
      </html>
   </body>
      <xsl:apply-templates/>
(<em><xsl:value-of select="author"/></em>)</h1>
   <h1><xsl:value-of select="title"/>
   <body>
<head><title><xsl:value-of select="title"/></title></head>
   <html>
<xsl:template match="/poem">
```

- Titel und Autor extrahieren:

Daten auswählen, cont.

- XPath-Achsen ausnutzen
- Navigationselemente:

```
previous slide:
<xsl:value-of select="preceding-sibling::slide[1]/title"/>,
next slide:
<xsl:value-of select="following-sibling::slide[1]/title"/>
```

- `preceding`/`preceding-sibling` liefern Knoten in umgekehrter Dokumentreihenfolge

template-Aufrufe steuern

- nächster Knoten wird verarbeitet

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

- nur bestimmte Knoten weiterverarbeiten:

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

- auch hier: XPath-Ausdruck

Aufgaben

- Schreibe das letzte Stylesheet so um, daß es ohne

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

  auskommt und stattdessen

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

  verwendet. Wieviel Code muß geändert werden? Woran liegt das?

- Welche Variante (`apply-templates` oder `value-of`) ist besser? Begründe Deine Wahl.

weitere Ausgabeverfahren

- Leerzeichen wird entfernt:

```
<xsl:value-of .../> <xsl:value-of .../>
```

- Textausgabe erzwingen:

```
<xsl:value-of .../><xsl:text> </xsl:text><xsl:value-of .../>
```

- Kommentar einfügen:

```
<xsl:comment>
   Autogenerated with pres2html.xsl. DO NOT MODIFY!!!
</xsl:comment>
```

weitere Ausgabeverfahren, cont.

- alternative Methode zur Element-Generierung:

```
<xsl:element name="ul">
  <xsl:apply-templates/>
</xsl:element>
```

- alternative Methode, um Attribut einzufügen

```
<xsl:element name="body">
  <xsl:attribute name="bgcolor">white</xsl:attribute>
  ...
</xsl:element>
```

- Mischform:

```
<body>
  <xsl:attribute name="bgcolor">white</xsl:attribute>
  ...
</body>
```

Aufgaben

- Schreibe ein Stylesheet `cd2html.xsl`, das die CD-Sammlung in `cd-collection.xml` in eine HTML-Datei umformt:

```
<h1>The Beatles / Help! (1965)</h1>
<ul>
  <li>Help!</li>
  <li>The Night Before</li>
  ...
</ul>

<h1>The Rolling Stones / Voodoo Lounge (1994)</h1>
...
```

- Füge eine Statistik auf der HTML-Seite ein, in der die Anzahl der CDs und die Gesamtzahl der Lieder angegeben wird.

```
<body>
  <p>
    I have
    <xsl:value-of select="count(/collection/cd)"/>
    CDs with a total of
    <xsl:value-of select="count(//song)"/>
    songs.
  </p>
  <xsl:apply-templates/>

<xsl:template match="cd">
  <h1>
    <xsl:value-of select="artist"/>
    <xsl:text> / </xsl:text>
    <xsl:value-of select="title"/>
    <xsl:text> (</xsl:text>
    <xsl:value-of select="@year"/>
    <xsl:text>)</xsl:text>
  </h1>
  <xsl:apply-templates select="songlist"/>
</xsl:template>
```