XML – Merkblatt 6 – XSLT

Einführung Die Extensible stylesheet transformation language (XSLT) ist eine Programmiersprache um XML-Dokumente (den Eingabe-XML-Baum) in andere XML- oder Text-Dokumente umzuwandeln (dem Ausgabe-XML-Baum). Um XSLT-Programme auszuführen zu können wird ein XSL-Prozessor, eine Art Interpreter, benötigt.

Beispiel Wir betrachten als Eingabe-XML ein fiktives Bug-Tracking-XML-Dokument.

```
<bug-tracking>
  <project name="HAL" version="90.0.0">
    <defect created="2001-08-15T19:05:10" level="1">
      <owner>Frank Poole</owner>
      <description>Increased neurotic system behaviour</description>
      <status>in progress</status>
    </defect>
    <defect created="2001-08-17T08:12:55" level="5">
      <owner>David Bowman</owner>
      <description>Malfunctional control of EVA pod</description>
      <status>open</status>
    </defect>
  </project>
  <project name="Multivac">
    <defect created="2061-06-30T15:15:15" level="1">
      <owner>Isaac Asimov</owner>
      <description>INSUFFICIENT DATA FOR MEANINGFUL ANSWER error</description>
      <status>open</status>
    </defect>
  </project>
</bug-tracking>
```

Das Wurzelelement `stylesheet` im folgenden Beispiel definiert den Namensraum für XSLT und enthält

- ein `xsl:output`-Element, mit dem die Art der Ausgabesyntax (erlaubt sind `xml` (Standardwert), `html` oder `text`) definiert wird, und
- zwei *Regeln* (template rules).

Eine Regel hat die Form `<template match="xpath">..</template>` Der XPath selektiert Knoten im XML-Baum relativ zum Kontextknoten. Wenn die resultierende Knotenmenge nicht leer ist, wird der Inhalt der Regel auf jeden Knoten angewendet. Der Kontext ändert sich dabei temporär zum jeweils angewendeten Knoten, bis die Regel beendet ist.

Nach Start des Programms, ist der Kontextknoten die Wurzel des Eingabedokuments. Die erste Regel im Beispiel hat einen nicht leeren Match und wird ausgeführt.

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html><head/>
      <body>
        <h1>Projects with open defects</h1>
        <xsl:apply-templates select="bug-tracking/project"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="project[defect/status = 'open']">
    Project <xsl:value-of select="@name"/>
  </xsl:template>
</xsl:stylesheet>
```

Jeder Text vor und nach XSLT-Befehlen sowie die Ausgabe, die ein XSLT-Befehl erzeugt, wird dabei in das Ausgabedokument ausgegeben.

`xsl:apply-templates` weist den XSL-Prozessor an, für alle Knoten des `select`-Attributs (optional, Standardwert `*`) rekursiv passende Regeln zu suchen und auszuführen. Im Beispiel wird für die Knotenmenge, die aus zwei `project`-Element besteht, die zweite Regel gefunden und für jeden Knoten (hier zwei Mal) ausgeführt. Bei jeder Ausführung wandert der Kontextknoten temporär zum angewendeten Knoten.

`xsl:value-of` fügt den Wert des mit `select` angegebene XPath als Textknoten in den Ausgabebaum ein.

Die erzeugt Ausgabe ist:

```
<html>
<head/>
<body>
<h1>Projects with open defects</h1>
  Project HAL
  Project Multivac</body>
</html>
```

Standard Templates Falls wir in unser Beispieldokument den Status des 2. Projekts auf `in progress` ändern, wird nach `Project HAL` auch folgender Text ausgegeben:

```
Isaac Asimov
INSUFFICIENT DATA FOR MEANINGFUL ANSWER error
in progress
```

Dieser Text wird nicht durch unsere angegeben Regeln erzeugt, sondern von Standardregeln, die je nach Konfiguration des verwendeten XSL-Prozessors implizit existieren können:

```
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="text()|@*">
  <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="processing-instruction()|comment()"/>
```

Bei Ausführung von `apply-templates select="bug-tracking/project"` wird für den zweiten `project`-Knoten, die erste Standardregel ausgeführt. Dies führt weitere Suchen nach sich, so dass schließlich jeder inneren Textknoten ausgeben wird.

Dieses Verhalten ist für Anfänger oft verwirrend und auch nicht immer gewünscht. Die Standardregeln lassen sich bei vielen XSL-Prozessoren ausschalten. Man kann sie auch im eigenen Programm überschreiben.

XML Ausgabe XSLT wird in der Regel verwendet um wieder XML-Dokumente zu erzeugen. Das Ausgabedokument hat oft eine anderen XML-Syntax als das Eingabedokument. Start- und End-Tags der Ausgabe können als Text angegeben.

Mit `xsl:text` können Textknoten erzeugt werden. Der Vorteil gegenüber dem einfachen Hinschreiben des Textes ist es, dass Leerzeichen außerhalb dieses Elements nicht in der Ausgabe erscheinen. Der Ausgabetext lässt sich deswegen gezielt formatieren, ohne das die verschachtelte Struktur der Regel dadurch leidet.

Mit `xsl:attribute name="attribut"` das mit `name` definierten Attribut innerhalb eines Start-Tags eingefügt. Als Attributwert wird der Inhalt von `xsl:attribute` genommen.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/bug-tracking">
    <project-members>
      <xsl:apply-templates select="project/defect/owner"/>
    </project-members>
  </xsl:template>
  <xsl:template match="owner">
    <member>
      <xsl:attribute name="project">
        <xsl:value-of select="../../../project/@name"/>
      </xsl:attribute>
      <xsl:text><xsl:value-of select="substring-after(.,' ')"/></xsl:text>
      <xsl:text>, </xsl:text>
      <xsl:text><xsl:value-of select="substring-before(.,' ')"/></xsl:text>
    </member>
  </xsl:template>
</xsl:stylesheet>
```

Das erzeugt XML ist:

```
<project-members>
<member project="HAL">Poole, Frank</member>
<member project="HAL">Bowman, David</member>
<member project="HAL">Asimov, Isaac</member>
</project-members>
```

Steuerungsanweisungen `xsl:if test="xpath"` führt den Inhalt nur aus, wenn der XPath zu `true` ausgewertet wird. `xsl:for select="xpath"` führt den Inhalt für jeden Knoten des XPath aus. Der Kontextknoten wandert bei jeder Ausführung temporär zum jeweiligen Knoten des XPaths.

Filter Bei einem *Filter* werden einige Inhalte der Eingabe weggelassen. Die Ausgabesyntax entspricht der Eingabesyntax. `xsl:copy select="xpath"` kopiert die mit select (Standardwert `.`) ausgewählten Elementknoten in die Ausgabe ohne den Inhalt auch Attribute dabei mitzukopieren (flache Kopie). `xsl:copy select="xpath"` kopiert den kompletten Inhalt der Knoten in die Ausgabe (tiefe Kopie). Die folgenden Transformation gibt alle Projekte mit den offenen defects aus:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/bug-tracking">
    <xsl:copy>
      <xsl:for-each select="project">
          <xsl:if test="defect[status='open']">
            <xsl:copy>
              <xsl:copy-of select="defect[status='open']"/>
            </xsl:copy>
          </xsl:if>
      </xsl:for-each>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Ergänzende Literatur Zur Einführung [1, Kapitel 17] (Das Kapitel ist online erhältlich).

Literatur

[1] Elliotte Rusty Harold. *XML Bible*. Wiley, 2nd edition, June 2001. http://www.cafeconleche.org/books/bible2/chapters/ch17.html.