Seminararbeit

XSLT

Nina Doschek (0815302)
Nina.Doschek@student.uibk.ac.at

18. Februar 2011

Betreuer: Dr. René Thiemann

Zusammenfassung (Englisch)

This seminar report gives an overview introduction to XSLT, which is a language for transforming XML documents. The syntax and semantics of XSLT are explained as well as the features of XSLT are shown in detail.

Inhaltsverzeichnis

1 Übersicht

In diesem Kapitel wird die Programmiersprache XSLT samt ihrem Hintergrund kurz vorgestellt und ihre Entwicklung behandelt.

1.1 Allgemeines

XSLT (= Extensible Stylesheet Language Transformation) ist eine deklarative, funktional-applikative Programmiersprache zur Transformation von XML-Dokumenten.

Deklarative Programmiersprachen sind meist funktionale/logische Sprachen und gehören der 5. Generation von Programmiersprachen an. Im Vordergrund steht das Beschreiben von Sachverhalten und Problemen, die Problemlösung wird durch das System vorgenommen. Für die deklarative Programmierung sprechen kleinere Programme. Die Beschreibung des Problems steht im Vordergrund, der Lsungsweg wird dann automatisch ermittelt.

Beispiele für weitere deklarative Programmiersprachen sind:

- Funktionale Programmiersprachen: OCaml, ML, Haskell, Scala
- Logische Programmiersprachen: Prolog
- Abfragesprachen: SQL

Die Funktionale Programmierung kennt keine Wertzuweisung. Bei diesem Programmierstil bestehen die Programme ausschließlich aus Funktionen. Um genauer zu sein, ist ein Programm eine Menge von Funktionen und transformiert die Eingabe in eine erwartete Ausgabe.

1.2 Geschichte und Entwicklung

XSLT wurde vom W3C (World Wide Web Consortium), dem Gremium zur Standardisierung des World Wide Web, im Jahr 1997 entwickelt. XSLT 2.0 ist die derzeit aktuelle Version, und wurde ursprünglich in den Jahren 2002-2006 entwickelt.

Vorgänger von XSLT war DSSL (Document Style Semantics and Specification Language). DSSL ist eine Transformationssprache für SGML-Dokumente. SGML (Standard Generalized Markup Language) wurde zu XML weiterentwickelt.

2 Einführung

Dieses Kapitel gibt eine kurze Einfühung in XML, stellt die Grundkonzepte und die Syntax von XSLT vor, hebt Merkmale hervor, geht auf die Anwendungsgebiete der Sprache ein und zeigt Unterschiede zu verwandten Sprachen auf.

2.1 XML-Dokumente

XML steht für Extensible Markup Lanugage [1] und dient zur Darstellung von hierarchisch strukturierten Daten und wird daher oft als Baumstruktur angesehen. XML dient außerdem zum Datenaustausch zwischen verschiedenen Computersystemen aufgrund von Plattform- und Implementationsunabhängigkeit. XML besitzt eine strenge Syntax, die dadurch aber eine einfache und effiziente Verarbeitung gewährleistet. Elemente werden innerhalb von Tags gespeichert.

Arten von Tags:

- Paar-Tags
  - `<tag-name> ... </tag-name>`
    Das gespeicherte Element wird zwischen zwei Tags eingetragen. zB ¡name¿Michael¡/name¿
- Empty-Element-Tags
  - `<tag-name />`
    Ein solches Tag wird meist verwendet, um die Struktur aufrecht zu erhalten, selbst wenn ein Element leer ist.

Angeben von Attributen:

- bei Paar-Tags
  - `<tag-name attribut-name=ättribut-Wert" > ... </tag-name>`
- bei Empty-Element-Tags
  - `<tag-name attribut-name=ättribut-Wert"/>`

Erstellen von Kommentaren:

- Kommentar
  - `<!-- Kommentar -->`

Beispiel zu XML:

```
<personen>
 <person>
   <titel />
   <vorname>Michael</vorname>
   <zuname>Kofler</zuname>
   <adresse>Innrain 45</adresse>
   <plz land="A">6020</plz>
   <ort>Innsbruck</ort>
   <geburtsdatum>
      <tag>05</tag>
      <monat>08</monat>
      <jahr>1980</jahr>
   </geburtsdatum>
 </person>
</personen>
```

2.2 Grundkonzepte von XSLT

Die Sprache XSLT [3] erzeugt durch Umwandlung eines XML-Dokuments ein neues Dokument. Das ursprüngliche XML-Dokument wird nicht verändert, es dient nur als Eingabe. Das Ausgabedokument ist meistens ein XML-Dokument (wie etwa ein HTML-Dokument), es ist jedoch auch möglich andere Dokumente wie zB PlainText- oder PDF-Dateien zu erstellen.

An einer erfolgreichen Transformation sind folgende Komponenten beteiligt [4], [5]:

- XML-Quell-Dokument(e) - die Eingabe
- XSLT-Stylesheet(s) - das Programm
- XSLT-Prozessor - der XSLT-Interpreter
- Ausgabe-Dokument(e)

2.2.1 Quell-Dokumente

Die Eingabedokumente sind XML-Dokumente, welche XSLT als logische Baumstruktur interpretiert. Das heißt, es gibt einen Wurzelknoten, von dem aus auf dessen Kinder zugegriffen werden können.

2.2.2 XSLT-Stylesheets

Ein XSLT-Stylesheet enthält die notwendigen Tranformationsregeln, um das Eingabedokument in die Ausgabe zu verwandeln. Solche Tranformationsregeln heißen "Templates". Auf Templates wird im Kapitel 2.3.1 näher eingegangen.

2.2.3 XSLT-Prozessor

Der XSLT-Prozesser stellt im Vergleich zu anderen Programmiersprachen den Interpreter bzw. Compiler dar.

2.2.4 Ausgabe-Dokumente

Die Ausgabe-Dokumente liegen meist bereits in HTML-Formaten vor und können sofort als solche angezeigt werden.

2.3 XSLT-Stylesheet Struktur

Ein XSLT-Stylesheet ist ein XML-Dokument, gekennzeichnet durch das `xsl:stylesheet` Element.

```
<xsl:stylesheet
  ...
  version = number>
  <!-- Content: (xsl:import*, top-level-elements) -->

</xsl:stylesheet>
```

Dieses Beispiel zeigt alle möglichen Attribute eines `xsl:stylesheet`-Elements. Als Synonym kann auch das Element `xsl:transform` verwendet werden. Das Attribut `version` ist als einziges Attribut verpflichtend und gibt die Version von XSLT an.

Innerhalb des Stylesheets können folgende top-level Elemente verwendet werden (Auswahl).

`xsl:import`	importiert ein weiteres XSLT-Stylesheet
`xsl:output`	definiert, wie der Ziel-Baum ausgegeben werden soll
`xsl:decimal-format`	formatiert Zahlen
`xsl:variable`	definiert eine Konstante in XSLT
`xsl:param`	definiert einen Parameter in XSLT (in anderen Sprachen vergleichbar mit einer Variable)
`xsl:template`	definiert eine Transformationsregel (Template)

2.3.1 XSLT-Templates

Ein XSLT-Template weist die folgende Struktur auf:

```
<xsl:template
  match = pattern
  name = qname ... >
  ...
</xsl:template>
```

Das match-Attribut liefert ein Pattern, das angibt, auf welchen Ausgangsknoten die Transformation des Templates angewandt wird. Weiters gibt es noch das name-Attribut das ein Template benennt. Wird ein Name vergeben, ist das match-Attribut nur mehr optional, andernfalls notwendig.

2.3.2 XSLT-Templates verwenden

Um mit einem Template eine Transformation durchzuführen, gibt es zwei Möglichkeiten, ein Template aufzurufen:

- `xsl:apply-template`
- `xsl:call-template`

apply-template

Wird das Element `apply-template` ohne weitere Optionen angewandt, werden alle Text- und Kind-Knoten des aktuellen Knotens ausgewählt. Auf diese Knoten werden dann die Transformationsregeln des darüberliegenden Templates angewandt.

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

Wird allerdings ein `select`-Attribut gesetzt, werden nur die Kind-Knoten ausgewählt und transformiert, die dem Ausdruck dieses Attributs entsprechen.

```
<xsl:apply-templates
   select = node-set-expression ... >
   ...
</xsl:apply-templates>
```

call-template

Mit dem Element `call-template` können benannte Templates aufgerufen werden, indem der Name des gewünschten Templates als Attribut übergeben wird. Beim Aufruf ist es möglich, diesem Template einen Parameter zu übergeben. Die Parameterübergabe wird in Kapitel 2.3.3 erklärt.

```
<xsl:call-template
     name = qname>
     ...
</xsl:call-template>
```

2.3.3 Parameter

In einem Template gibt es die Möglichkeit Parameter zu definieren, und ihnen einen default-Wert zuzuweisen.

```
<xsl:param
  name = qname
  select = expression>
  ...
</xsl:param>
```

Die Parameterübergabe wird mit `xsl:with-param` ermöglicht. Der Wert des Parameter wird mit dem `select`-Attribut festgelegt. Wird ein Wert übergeben, wird ein etwaiger default-Wert des Parameter ignoriert.

```
<xsl:with-param
  name = qname
  select = expression>
  ...
</xsl:with-param>
```

Beispiel

Im folgenden Beispiel wird gezeigt, wie die Rekursion in XSLT realisiert wird.

```
<xsl:template name="dots">
 <xsl:param name="count" select="1" />
 <xsl:if test="$count &gt; 0">
  <xsl:text>.</xsl:text>
  <xsl:call-template name="dots">
    <xsl:with-param name="count"
        select="($count)-1" />
  </xsl:call-template>
 </xsl:if>
</xsl:template>
...
<xsl:call-template name="dots">
 <xsl:with-param name="count" select="25" />
</xsl:call-template>
```

In diesem Beispiel wird ein Template mit dem Namen "dots" erstellt. Anschließend wird ein Parameter `count` mit dem default-Wert 1 definiert. Der Wert dieses Parameters ("`$count`") wird darauf überprüft, ob er größer als 0 ist. Wenn dem so ist, wird zuerst ein Punkt ausgegeben und anschließend wird das Template rekursiv aufgerufen. Dem Template der um eins verminderte aktuelle Parameter-Wert übergeben.
Später im Programm kann das Template mit `call-template` und der Übergabe eines Werts aufgerufen werden.

2.3.4 XPath

XPath [2] ist eine ebenfalls vom W3C entwickelte Sprache, die eine einfache Auswahl von Knoten eines XML-Dokuments ermöglicht. Eine XPath-Expression besteht aus der Auswahl der Knoten, die recht intuitiv ist, da sie dem Filesystem eines Computers ähnelt. Zusätzlich ist es noch möglich Prädikate anzugeben, um die Auswahl der Knoten zu filtern.

Auswahl der Knoten:

Knotenname	alle Kinderknoten dieses Knotens
/	Wurzelknoten
//Knotenname	Alle Knoten im Dokument, die diesem Namen entsprechen
.	aktueller Knoten
..	Vorgänger des aktuellen Knotens
@	Attribut eines Knotens

Filtern mittels Prädikaten:

Knotenname[1]	Auswahl der ersten Kindknoten
Knotenname[last()]	Auswahl der letzten Kindknoten
Knotenname[@attr]	Auswahl der Knoten mit dem Attribut attr
Knotenname[Kind > Wert] Knotenname[Kind < Wert] Knotenname[Kind = Wert]	Auswahl der Knoten, dessen Kinder größer/kleiner/gleich eines bestimmten Wertes sind

XPath Beispiel (XML-Grundlage: Personen-Beispiel aus Kapitel 2.1)

XPath-Expression	**Ergebnis**
/personen/person[1]	wählt die erste Person im personen-Datensatz aus
/personen/person[last()]	wählt die letzte Person im personen-Datensatz aus
//plz[@land]	wählt alle Postleitzahlen aus, die ein Attribut "land" besitzen
//plz[@land='A']	wählt alle Postleitzahlen aus Österreich aus
//geburtsdatum[jahr>1980]	wählt die Geburtsdaten nach 1980 aus

2.4 Kontrollstrukturen in XSLT

2.4.1 for-each

Die for-each Schleife besitzt ein `select`-Attribut, das angegeben werden muss, um die Knoten auszuwählen, auf welche der Inhalt der Schleife angewandt werden soll.

```
<xsl:for-each select=node-set-expression>
  ...
</xsl:for-each>

<!-- Beispiel -->
<xsl:for-each select="personen/person">
  <xsl:apply-templates select="zuname"/>
  <xsl:text> </xsl:text>
  <xsl:for-each select="vorname">
    <xsl:apply-templates/>
    <xsl:text> </xsl:text>
  </xsl:for-each>
</xsl:for-each>
```

In diesem Beispiel werden alle Vornamen einer Person ausgegeben, getrennt durch ein Leerzeichen. Die for-each-Schleife ist daher sinnvoll, da es von der jeweiligen Person abhängig ist, wieviele Vornamen sie hat.

2.4.2 if-Anweisung

Mit einer if-Anweisung in XSLT können nur binäre Entscheidungen getroffen werden, was einen Unterschied zu den meisten Programmiersprachen darstellt.

```
<xsl:if test = boolean-expression>
  ...
</xsl:if>

<!-- Beispiel -->
<xsl:template match="personen/person">
  <xsl:if test="plz/@land='A'">AUT</xsl:if>
</xsl:template>
```

In diesem Beispiel wird für alle Personen "AUT" ausgegeben, die in Österreich wohnen.

2.4.3 choose-Konstrukt

Mit diesem Konstrukt sind mehrfache Entscheidungen möglich. Es ist mindestens ein `when`-Element notwendig, das `otherwise`-Element ist optional.

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

```
<xsl:when test=boolean-expression>
  ...
</xsl:when>

<xsl:otherwise>
  ...
</xsl:otherwise>

<!-- Beispiel -->
<xsl:template match="personen/person">
  <xsl:choose>
    <xsl:when test="ort='Innsbruck'">
      IBK
    </xsl:when>
    <xsl:when test="ort='Salzburg'">
      SBG
    </xsl:when>
    <xsl:otherwise>
      N/A
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

In diesem Beispiel werden 2 Fallunterscheidungen vorgenommen: "IBK" wird ausgegeben, wenn eine Person in Innsbruck wohnt, "SBG" wird ausgegeben, wenn eine Person in Salzburg wohnt und falls eine Person in keiner dieser beiden Städte wohnt, wird "N/A" ausgegeben. Mit diesem choose-Konstrukt kann eine if-then-else-Verzweigung realisiert werden. In diesem Fall wird eine verschachtelte Verzweigung simuliert.

3 Ausblick

In diesem Kapitel werden die Anwendungsgebiete von XSLT, die Merkmale, Besonderheiten und Nachteile von XSLT besprochen. Außerdem werden geeignete Sprachen angeführt, durch die XSLT ersetzt werden könnte.

3.1 Anwendung von XSLT

- Presentation Oriented Publishing [4]: Die Transformation dient zum Zwecke der Darstellung.
  Mit Stylesheets ist es möglich, XML-Daten in viele verschiedene Formate (wie zB Vektorgrafiken, Text, Tabellen in XHTML, PDF-Dateien, Excel-Tabellen) zu transformieren.

- Message Oriented Middleware [4]: Die Transformation dient zum Zwecke des Datenaustausches.
  Es ist nicht zwingend notwendig, dass Systeme, die miteinander kommunizieren wollen, die gleiche XML-basierte Sprache verwenden. Daher kann XSLT eingesetzt werden, um die Transformation von einer in die andere Sprache durchzuführen.

3.2 Merkmale und Besonderheiten von XSLT

Das größte Plus von XSLT ist die schnelle Transformation von XML-Daten in HTML-Dokumente. Außerdem wird bei der Transformation gewährleistet, dass die Dokumente wohlgeformt sind. Zu den Vorteilen von XSLT zählt auch die mögliche Verwendung von XPath, da dadurch die Auswahl der zu transformierenden Knoten um vieles erleichtert wird.

3.3 Nachteile von XSLT

Die Wartung von XSLT-Stylesheets gestaltet sich schwierig, da die Syntax nicht sehr intuitiv und der Code dadurch schnell unübersichtlich erscheint. Im Vergleich zu anderen funktionalen Programmiersprachen ist es nicht möglich, Funktionen höherer Ordnung zu erstellen. Funktionen höherer Ordnung erhalten Funktionen als Argumente oder liefern Funktionen als Ergebnis zurück. (Vergleich: map-Funktion in OCaml) Außerdem ist es nicht möglich, während der Laufzeit ein XML-Dokument zu erstellen und dieses intern weiterzuverarbeiten.

3.4 Alternativen zu XSLT

- DSSSL
  DSSSL ist der Vorgänger von XSLT und daher nicht mehr geeignet, um XSLT zu ersetzen.

- Generische Programmiersprachen wie Java oder C++
  Mit diesen Programmiersprachen könnten die Transformationen sehr wohl vorgenommen werden, aber es besteht keine direkte Möglichkeit, sofort HTML-Dokumente zu erstellen.

- CSS
  CSS wird zwar oft als Alternative zu XSLT genannt, aber diese Sprache könnte XSLT nie im gesamten Umfang ersetzen. CSS ist eine reine Formatierungssprache und es können daher keine Transformationen vorgenommen werden.

4 Zusammenfassung

XSLT ist eine deklarative, funktionale, XML-basierte Sprache zur Transformation von XML-Dokumenten. Das Eingabe-Dokument wird nicht verändert, es wird ein neues Dokument erstellt, das auf dem Inhalt des Eingabe-Dokuments basiert. Mit XSLT wird es möglich, schnell XML-Daten als HTML zu repräsentieren. XSLT liefert dazu zusätzlich noch die Merkmale einer funktionalen Sprache, wie zB die effektive Verwendung der Rekursion. XSLT ist verfügbar in Browsern, was die Sprache auch aktuell relevant macht.

Literatur

[1] W3C. Extensible markup language (xml) 1.0 (fifth edition), January 2011. http://www.w3.org/TR/xml/.

[2] W3C. Xml path language (xpath), January 2011. http://www.w3.org/TR/xpath/.

[3] W3C. Xsl transformations (xslt), January 2011. http://www.w3.org/TR/xslt.

[4] wikipedia.org. Xsl transformation, January 2011. http://de.wikipedia.org/wiki/Xslt.

[5] wikipedia.org. Xslt, January 2011. http://en.wikipedia.org/wiki/Xslt.