XSLT, Teil 3

Udo Kelter

17.06.2009

Zusammenfassung dieses Lehrmoduls

Auch in XML-Datenbeständen hat man das Problem der Verbundbildung, also der Bildung neuer Elemente, die Daten aus verschiedenen Quellen zusammenführen. In XSLT müssen solche Verbunde praktisch von Hand berechnet werden; hierzu benötigt man Variablen oder manuell verwaltete Sekundärindexe. Dieses Lehrmodul stellt das weitere XSLT-Kommandos, das Variablenkonzept und Sekundärindexe vor.

Vorausgesetzte Lehrmodule:

obligatorisch:
- XPath
- XSLT, Teil 1 (Stichworte)
- XSLT, Teil 2 (Stichworte)

Stoffumfang in Vorlesungsdoppelstunden: 1.0

Inhaltsverzeichnis

1 Ausgabe- und Steuerkommandos

hier nur punktuell und informell dargestellt, Details (insb. weitere Parameter) und genaue Definition s. XSLT-Standard

Gemeinsamkeiten:

- sind entweder Steuerkommandos oder erzeugen direkt irgendwelche Teile des Ausgabebaums
- treten in Schablonen auf
- haben oft einen Inhalt, der wiederum eine Schablone darstellt, m.a.W. können Ausgabekommandos geschachtelt werden
- werden als Element mit Typ `xsl:.....` notiert

1.1 Versuch einer Systematik und Übersicht

1. Steuerkommandos

   `xsl:for-each` iteriert über eine Knotenmenge und ruft für jeden Knoten die innere Schablone auf

   `xsl:apply-templates` iteriert über eine Knotenmenge und wendet die jeweils zutreffende Transformationsregel an

   `xsl:if` bedingter Aufruf einer inneren Schablone

   `xsl:choose` verallgemeinertes `if`; mehrere nacheinander zu testende Bedingungen und ggf. eine `otherwise`-Alternative

2. Ausgabekommandos, die *Elementknoten* erzeugen und den Elementbaum aufbauen:

   `<...>` direkte Angabe der öffnenden und schließenden *tags*

   `xsl:copy` kopiert einen Knoten des Eingabebaums; Typ des Knotens ist beliebig

   `xsl:element` erzeugt einen Elementknoten; der Typ wird als Parameter angegeben und kann dynamisch berechnet werden

3. Ausgabekommandos, die *Textknoten* erzeugen

`...`	direkte Angabe von Text
`xsl:text`	erzeugt den angegebenen Text; spezielle Möglichkeiten zur Behandlung von Leerraum
`xsl:value-of`	konvertiert einzelne Eingabeknoten oder ganze Teilbäume in textuelle Darstellung

4. Ausgabekommandos, die *Attributknoten* erzeugen

`xxx='...'`	direkte Angabe von Attributname und Wert im öffnenden *tags*; nur möglich, wenn auch das Element direkt angegeben wird
`xsl:attribute`	erzeugt einen Attributknoten; Attributname und Wert werden in Parametern angegeben und können dynamisch berechnet werden
`xsl:copy`	kopiert auch Attributknoten des Eingabebaums

1.2 Das Kommando `xsl:if`

Schema:

```
<xsl:if test = 'boolean-expression' >
  <!-- innere Schablone -->
</xsl:if>
```

Merkmale:

- hat einen Parameter `test`, der einen Booleschen Ausdruck enthält (Vorsicht mit `<` -Zeichen! müssen umcodiert / vermieden werden)
- Inhalt: innere Schablone
- kein "else"-Zweig möglich

Wirkung:

1. der im Parameter `test` enthaltene Booleschen Ausdruck wird ausgewertet
2. bei positivem Testergebnis wird die innere Schablone ausgeführt
   Kontextknoten für die innere Schablone: der gleiche (!) wie der der aufrufenden Schablone

1.3 Das Kommando `xsl:choose`

entspricht einer case- / switch-Verzweigung
Schema:

```
<xsl:choose>
   <xsl:when  test = 'boolean-expression' >
     <!-- Content: template -->
   </xsl:when>
   ......
   ......
   <xsl:otherwise>
     <!-- Content: template -->
   </xsl:otherwise>
</xsl:choose>
```

1.4 Attributwertschablonen

Problem, falls Attributwerte keinen festen Wert erhalten, sondern *berechnet* werden sollen:

- keine inneren Elemente erlaubt
- daher Ausgabekommandos in der Notation als `xsl:...`-Element nicht direkt im Inhalt eines Attributs erlaubt

direkte Angabe des Werts nur brauchbar, wenn Wert immer gleich (ggf. abhängig von `xsl:if`)

Beispiel: die Telefonliste aus LM XSLT2 soll in folgende Form umgewandelt werden:

```
<Telefonliste>
  <Eintrag name="Meier" laendercode="0049" vorwahl="0271"
    nummer="891234"/>
  <Eintrag name="Schmitz" laendercode="0049" vorwahl="0228"
    nummer="870887"/>
</Telefonliste>
```

Attribut `laendercode` kann direkt mit festem Wert angegeben werden:

```
  <xsl:template match='Eintrag'>
    <Eintrag laendercode="0049" vorwahl="??????" ....  />
  </xsl:template>
```

Die Werte der anderen Attribute hängen von anderen Knoten des Eingabebaums ab;

z.B. Wert von `nummer` ist Kopie des Inhalts des Elements `Telnr`

`xsl:value-of` kann man nicht benutzen:

```
  <xsl:template match='Eintrag'> <!-- FALSCH !!!!! -->
    <Eintrag
      vorwahl="<xsl:value-of select='Telnr/@vorwahl' />"
      nummer ="<xsl:value-of select='Telnr' />"
    />
  </xsl:template>
```

ist falsch, weil `<` in Attributwerten nicht erlaubt ist und generell Attribute keine Kindelemente haben können!

(umcodieren des `<` mit `<` nützt nichts, weil das `<` dann "wörtlich" genommen wird und kein Kommando interpretiert wird)

Attributwertschablonen:

Attributwertschablone = Ausdruck, der Knotenliste liefert, in geschweiften Klammern

Syntax: { *Ausdruck*}

Ausgabe: textuelle Darstellung des *ersten* (!) Knotens der Liste

Beispiel:

```
<xsl:template match='Eintrag'>
  <Eintrag name   ="{name}"
           laendercode="0049"
           vorwahl="{Telnr/@vorwahl}"
           nummer ="{Telnr}" />
</xsl:template>
```

Beispiel 2: nur 1 Attribut mit kompletter Telefonnummer gemäß Muster "[0049] 0271-7402611"

```
<xsl:template match='Eintrag'>
  <Eintrag name   ="{name}"
           telefonnr ="[0049] {Telnr/@vorwahl}-{Telnr}" />
</xsl:template>
```

d.h. Attributwert wird durch *mehrere* Attributwertschablonen und feste Texte erzeugt

1.5 Das Kommando `xsl:attribute`

Anzuwenden, wenn auch *der Name des auszugebenden Attributs berechnet* werden soll oder wenn der Attributwert ziemlich komplex ist

Merkmale und Wirkung:

- `xsl:attribute`-Anweisung muß *vor* Anweisungen ausgeführt werden, die den Inhalt des Elements erzeugen
- hat einen Parameter `name`, der den Namen des zu erzeugenden Attributs angibt
- Inhalt: Wert des Attributs

Beispiel: In einer Lehrveranstaltungsbeschreibung alle Durchführungen in einem einzigen Attribut zusammenfassen;
Bsp: `<DURCHF.. semester='Meier:2007s; Koch:2008w;' />`

Lösung:

```
<xsl:template match=" DURCHFUEHRUNG " />
<xsl:template match=" DURCHFUEHRUNG[1] " >
  <DURCHFUEHRUNGEN>
    <xsl:attribute name="semester">
      <xsl:for-each select='../DURCHFUEHRUNG' >
        <xsl:value-of select='@dozentId' />:
        <xsl:value-of select='@semester' />;
      </xsl:for-each>
    </xsl:attribute>
  </DURCHFUEHRUNGEN>
</xsl:template>
```

1.6 Das Kommando `xsl:element`

Anzuwenden, wenn auch *der Name des auszugebenden Elements berechnet* werden soll

Merkmale und Wirkung:

- `xsl:element`-Anweisung statt öffnendem und schließenden Tag
- Parameter `name` gibt Namen des zu erzeugenden Elements an
- Parameter `namespace` deklariert Namensraumbezeichner für dieses Element
- Inhalt: Schablone, die Kinder des Elements erzeugt

Beispiel (um des Beispiels willen): DURCHFUEHRUNG-Elemente bilden, die die dozentId im Elementnamen enthalten

```
<xsl:template match=" DURCHFUEHRUNG " >
  <xsl:element
       name = 'DURCHFUEHRUNGvon{@dozentId}' >
    <xsl:attribute name="semester">
      <xsl:value-of select='@semester' />
    </xsl:attribute>
  </xsl:element>
</xsl:template>
```


2 Variablen

- Variable = im Prinzip Paar (Name, Wert)
- werden u.a. benötigt, um Gruppierungen / Aggregationen und Verbunde zu berechnen
- können in verschiedenen Kontexten benutzt werden (Suchbedingungen in Pfaden, Ausgabeanweisungen usw.)
- sehr viele Sonderfälle und Verhaltensvarianten → eher schlecht verständlich
- in XSLT Version 2 deutlich besser als in XSLT Version 1
- sind als top-level-Element zulässig, aber auch als Anweisung innerhalb von Schablonen

Deklaration:

```
<xsl:variable name = '...'
              select = '...' >
  <!-- Content: template -->
</xsl:variable>
```

- Attribut `name` muß syntaktisch korrekten Namen enthalten
- Angabe des Werts *entweder* im Attribut `select` *oder* durch eine innere Schablone

Benutzung in der Form $*variablenname*

2.1 Wertangabe in `select`

`select` muß Ausdruck enthalten, der eine Zeichenkette, eine Zahl, einen Boolschen Wert oder eine Knotenmenge liefert

falls Knotenmenge: Knoten können als Ausgangspunkt von Navigationen dienen; Beispiele:

```
<xsl:value-of select='$variablenname/lokalerPfad' />
<element attribut='{$variablenname/lokalerPfad}' />
```


Vorsicht: viele automatische Konversionen!
Beispiele:

```
<xsl:variable name="n1" select="2"/>
<xsl:variable name="n2" select="$n1+3*4"/>
<xsl:variable name="n3" select="'xxyyzz'"/>
<xsl:variable name="n4" select="''"/>
<xsl:variable name="n5" select="'2'"/>
<xsl:variable name="n6" select="1>2"/>
<xsl:variable name="n7" select="//@semester"/>

<xsl:template match=" / ">    <out n1="{$n1}" n2="{$n2}"
  n3="{$n3}" n4="{$n4}" n5="{$n5}" n6="{$n6}"  />
</xsl:template>
```

liefert:

```
<out n1="2" n2="14" n3="xxyyzz" n4="" n5="2" n6="false" />
```

2.2 Wertangabe in innerer Schablone

Beispiele:

```
<xsl:variable name="n8" >
  Dies ist ein <b>fetter</b> Text.
</xsl:variable>

<xsl:variable name="n9" >
  <xsl:for-each select="//@semester" >
    <xsl:value-of select='.' /><xsl:text>..</xsl:text>
  </xsl:for-each>
</xsl:variable>
```

Ergebnistyp: *result tree fragment*

Nutzungsmöglichkeiten eines *result tree fragments*:

- Ausgabe mit `xsl:value-of`: konvertiert zu einem Textknoten
- Ausgabe mit `<xsl:copy-of select='$xxx' />`: als komplette Kopie mit allen inneren Knoten

- können *nicht* als weitere Eingabe, Startpunkt von Navigationen o.ä. benutzt werden (in XSLT 1.0)

2.3 Verbundbildung mit Variablen

- Nachimplementierung eines Verbunds von Hand unter Benutzung von Variablen
- umständlich und oft sehr schlecht durchschaubar

Beispiel:

- wie bisher Lehrveranstaltungsdaten, die eine `dozentId` als "Fremdschlüssel" auf die Personendaten enthalten
- zusätzlich Personendaten

```
<FBINFO>
  <PERSONEN>
    <PERSON persId="Kelter" nachname="Kelter"
            vornameInit="U." fachgr="PI"      />
    ....
  </PERSONEN>

  <LEHRVERANSTALTUNG>
    ....
    <VERANTWORTLICHER dozentId="Kelter" />
    ....
  </LEHRVERANSTALTUNG>
</FBINFO>
```

Aufgabe: Im Element `VERANTWORTLICHER` sollen innen der Name, Initialien und Fachgruppenzugehörigkeit eingetragen werden

Lösungsstrategie: Verbund manuell implementieren

1. eine Variable mit dem Fremdschlüsselwert anlegen (`dzId`)
2. die Variable nutzen, um in der "Zielrelation des Fremdschlüssels" den zugehörigen Eintrag zu lokalisieren –
   in zweiter Variable (`dzRef`) Referenz auf diesen Eintrag speichern

3. von der Referenz in der zweiten Variablen aus zu den auszugebenden Daten navigieren

Lösungsausschnitt:

```
<xsl:template match=" VERANTWORTLICHER " >
  <!-- Schritt 1 -->
  <xsl:variable name='dzId' select='@dozentId' />
  <!-- Schritt 2 -->
  <xsl:variable name='dzRef'
                select='//PERSON [ @persId = $dzId ]' />
  <!-- Schritt 3 -->
  <VERANTWORTLICHER dozentId='{$dzId}' >
    <xsl:value-of select='$dzRef/@nachname' />,
    <xsl:value-of select='$dzRef/@vornameInit' />
    (<xsl:value-of select='$dzRef/@fachgr' />)
  </VERANTWORTLICHER>
</xsl:template>
```

Anmerkungen:

- `dozentId='@dozentId'` wäre ebenfalls richtig
- `dozentId='$dzId'` wäre falsch, würde `$dzId` wörtlich ausgeben
- `dzRef` enthält Menge von Referenzen auf Knoten im Eingabebaum, weil mit `select='`*pfad*`'` gesetzt;
  wenn Daten korrekt, max. 1 Element.
- von dort aus weiternavigieren,
  Beispiel: `$dzRef/@nachname`

3 Sekundärindexe

allgemein: Sekundärindex = Index, der einem Sekundärschlüsselwert eine Trefferliste zuordnet

“Trefferliste” im Kontext von XSLT: Menge von Knoten des Eingabebaums

3.1 Anlegen eines Sekundärindexes

XSLT-Kommando `xsl:key`

```
<!-- Category: top-level-element -->
<xsl:key  name = 'qname'
          match = 'pattern'
          use = 'expression' />
```

- Parameter `name`: Name des Sekundärindex;
  es können beliebig viele angelegt werden, die später durch ihren Namen identifiziert werden
- Parameter `match`: Typen der Knoten, die indexiert werden sollen (wie Parameter `match` in Transformationsregeln)
- Parameter `use`: Ausdruck, der zu einem Knoten dessen Sekundärschlüsselwert berechnet

Effekt der Ausführung eines `xsl:key`-Kommandos:

1. bestimme alle Knoten im Eingabebaum, die zu `match` passen
2. für jeden dieser Knoten: berechne den zug. Sekundärschlüsselwert gem. Parameter `use`

   kann aus mehreren Datenwerten mit Textfunktionen wie `concat(..)`, `substring(...)` usw. konstruiert werden

   Textfunktionen: s. XPath-Standard, 4.2 String Functions
3. bestimme zu jedem aufgetretenen Sekundärschlüsselwert die Trefferliste, also die Liste der Knoten mit diesem Sekundärschlüsselwert

3.2 Benutzen eines Sekundärindexes

Abruf der Trefferliste für einen Sekundärschlüsselwert mit der Funktion:

```
key ( SIName: string, SSWert: object ) :  node-set
```

- 1. Parameter: Bezeichner des Sekundärindexes

- 2. Parameter: Sekundärschlüsselwert
- Rückgabe: Trefferliste, also Liste von Referenzen auf Knoten des Eingabebaums

Funktion `key` hätte besser `getNodesForKeyValue` o.ä. geheißen

Beispiel 1: zeige alle Lehrveranstaltungen in einem bestimmten Semester an:

```
<xsl:key name="LVproSemester"
         match="DURCHFUEHRUNG"
         use="@semester" />

<xsl:template match=" / ">
  <xsl:for-each
         select=' key( "LVproSemester", "2006s" ) ' >
    <xsl:value-of select='@semester' />
    <xsl:value-of select='@dozentId' />
    <xsl:value-of select='../LEHRVERANSTALTUNGSNAME' />
  </xsl:for-each>
</xsl:template>
```

3.3 Verbundbildung mit Index

Beispiel 2: wie oben (Verbundbildung mit Variablen)

```
<xsl:key name="personendaten"
         match="PERSON"
         use="@persId" />
<xsl:template match=" VERANTWORTLICHER " >
  <VERANTWORTLICHER dozentId='{@dozentId}' >
    <xsl:for-each
         select=' key( "personendaten", @dozentId ) ' >
      <xsl:value-of select='@nachname' />,
      <xsl:value-of select='@vornameInit' />
      (<xsl:value-of select='@fachgr' />)
    </xsl:for-each>
  </VERANTWORTLICHER>
</xsl:template>
```

3.4 Gruppierung und Aggregation mit Index

Beispiele mit Zählung / Summierung:

```
Zahl der Module:
<xsl:value-of select=
              ' count(key("LVproSemester", "2006s")) ' />

Gesamtzahl der LP:
<xsl:value-of select=
           ' sum(key("LVproSemester", "2006s") / .. /
             LEISTUNGSPUNKTE / @anzahl) '              />
```

3.5 Bewertung von Indexen

- wesentlich effizienter als viele direkte Abfragen, wenn der gleiche Datenbestand wiederholt durchsucht werden muß
- günstig für Verbundbildung oder Gruppierung / Aggregation
- unverständliche Namen von Kommando / Funktion

4 Softwaretechnische Beurteilung von XSLT

- Lesbarkeit / linguistische Qualität der XSLT-Programme: desaströs, Fehlerquelle erster Güte

  ... man könnte auch Java-Programme als Syntaxbaum in XML codiert darstellen!
- kein vernünftiges Modulkonzept
- ungeeignet für komplexere Algorithmen / Applikationen → beschränken auf reine Abfragezwecke

  sinnvolle Trennung zwischen Datenextraktion – Fachlogik (in richtiger Programmiersprache!) anstreben
- Vorteil von XSLT bei sehr einfachen Applikationen: nur 1 Sprache

  Indiz, daß pures XSLT sinnvoll ist: man kommt mit wenigen “kleinen” Transformationen (50 - 200 Zeilen) aus

Literatur

[XPAT] Kelter, U.: Lehrmodul “XPATH”; 2009
[XSLT] Kelter, U.: Lehrmodul “XSLT, Teil 1 (Stichworte)”; 2009
[XSLT2] Kelter, U.: Lehrmodul “XSLT, Teil 2 (Stichworte)”; 2009

