XSLT, Teil 2 (Stichworte)

Udo Kelter

23.05.2007

Zusammenfassung dieses Lehrmoduls

Transformationsregeln sind das zentrale Konzept von XSLT. Motiviert sind Transformationsregeln durch die Annahme, daß i.d.R. der komplette Eingabebaum durchlaufen wird und man die gesamte Dokumenttransformation in lokale Anteile zerlegen kann, die dem Typ der Knoten des Eingabebaums zugeordnet werden können. Dieses Lehrmodul erklärt im Detail, wie Transformationsregeln spezifiziert und aufgerufen werden und welche Transformationsregeln vordefiniert sind.

Vorausgesetzte Lehrmodule:

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

empfohlen:
- XML-Namensräume

Stoffumfang in Vorlesungsdoppelstunden: 1.0

Inhaltsverzeichnis

1 Transformationsregeln

1.1 Transformationsregeln – Motivation

einfaches Beispiel einer "XML-Abfrage":

- gegeben sei eine XML-Datei, die eine Adreßliste enthält (wie in Lehrmodul XML)
- gesucht: "Projektion" auf Nachname und Telefonnummer, genauer gesagt:
  - i.w. gleicher Syntaxbaum
  - viele Knoten müssen identisch kopiert werden
  - manche Knoten / Teilbäume werden weggelassen
  - manche Knoten werden umbenannt

resultierende **Anforderungen an eine XML-Abfragesprache:**

1. Wiederholung: der *komplette Ausgabebaum muß in 1 Verarbeitungsschritt aufgebaut werden*
2. es sollte nicht nötig sein, den Durchlauf durch den Eingabebaum manuell zu programmieren
   (wäre immer wieder das gleiche Hauptprogramm)
   möglichst nur Abweichungen vom Standarddurchlauf angeben
3. das Kopieren von Teilbäumen / Baumfragmenten sollte einfach sein / wenig Schreibaufwand verursachen

1.2 Bisheriger Lehrstoff

- Transformationsregel für Dokumentwurzel, impliziter Aufruf dieser Transformationsregel bei Programmstart
- Kontext beim Aufruf einer Transformationsregel
- Ausführung einer Schablone (= Rumpf der Transformationsregel)
- Kommandos `value-of`, `text` und `for-each`
- `for-each`-Kommando hat innere Schablone; diese wird mehrfach in verschiedenen Kontexten ausgeführt

1.3 Neuer Lehrstoff - Übersicht

- Definition und Verwaltung vieler Transformationsregeln
- expliziter Aufruf von Transformationsregeln
- Priorität von Transformationsregeln
- vordefinierte Transformationsregeln
- identische Transformation und Projektionen

2 Definition und Verwaltung von Transformationsregeln

2.1 Spezifikation der Knotentypen im Parameter `match`

konzeptueller Inhalt einer Transformationsregel:

- Inhalt des `template`-Elements: Schablone / "Textmuster", das für einen Knoten dieses Typs auszugeben ist

  enthält Ausgabee- und Steuerkommandos, darf *keine inneren Transformationsregeln* enthalten

- Parameter `match`: Spezifikation, auf welche Knotentypen die Regel anwendbar ist

  einfachster und häufigster Fall: genau ein Knotentyp; alle Knotentypen separat wählbar

  i.a.: Spezifikation einer *mehrerer* Knotentypen

  ferner Einschränkung auf Knoten, die in einem bestimmten Kontext stehen

Parameter `match` – Spezifikation einzelner Knotentypen: identisch mit Knotentest in Navigationsschritten eines XPath-Pfads!

Syntax hängt von der Kategorie des jeweiligen Knotentyps ab:

Wurzelknoten: `match='/'`

Elemente: `match='`*Elementtypname*`'`

Attribute: `match='@`*Attributname*`'`

Textabschnitte: `match='text()'`

Kommentare: `match='comment()'`

Processing Instructions: `match='processing-instruction()'`

Parameter `match` – Spezifikation mehrerer Knotentypen mit Namensmustern:

für Elementtypen und Attribute: "Namensmuster"
identisch mit Namensmustern in Knotentest in Navigationsschritten eines XPath-Pfads! s. XPath-Spezifikation:

```
[37] NameTest ::= '*'  |  NCName ':' '*'  |  QName
```

s. XML-Namespace-Spezifikation:

```
[4]  NCName        ::=  NCNameStartChar NCNameChar*
                        /* An XML Name, minus the ":" */
                        /* (Non-Colon-Name) */
[5]  NCNameChar    ::=  NameChar - ':'
[6]  NCNameStartChar ::= Letter | '_'
[7]  QName         ::=  PrefixedName | UnprefixedName
[8]  PrefixedName  ::=  Prefix ':' LocalPart
[9]  UnprefixedName ::=   LocalPart
[10] Prefix        ::=  NCName
[11] LocalPart     ::=  NCName
```

keine beliebigen regulären Ausdrücke über Typnamen möglich!! letztlich nur folgendes erlaubt:

`*` selektiert alle Elemente

`NR:*` selektiert alle Elemente, deren Typname in dem Namensraum liegt, der durch den Namensraumbezeichner `NR` angegeben wird

`@*` selektiert alle Attribute

`@NR:*` selektiert alle Attribute, deren Name in dem Namensraum liegt, der durch den Namensraumbezeichner `NR` angegeben wird

Parameter `match` – Spezifikation mehrerer Knotentypen durch Aufzählung:

Angabe durch Aufzählung mit Trennzeichen `|`:

Beispiel: `match='Name | Vorname | @* | meinNR:*'`

allgemeine Syntax von Patterns = Inhalt des Parameters `match`:

```
[1] Pattern ::=
        LocationPathPattern
        | Pattern '|' LocationPathPattern
```

Reihenfolge der `LocationPathPattern` in einem `Pattern` ist unerheblich

Parameter `match` – LocationPathPatterns:

zusätzliche Selektion nach der typbasierten Selektion: Knoten müssen "im Kontext" anderer Knoten stehen;
Beispiele:

`ol/item` selektiert alle `item`-Elemente, deren Elternknoten ein `ol`-Element ist

`DURCHFUEHRUNG/@dozentId` selektiert alle `dozentId`-Attribute, deren Elternknoten ein `DURCHFUEHRUNG`-Element ist

allgemeine Form von *LocationPathPattern*s:

- sind "nach unten gehende" XPath-Ausdrücke

allgemeine Syntax von LocationPathPatterns:


```
[2] LocationPathPattern ::=
        '/' RelativePathPattern?
        | '//'? RelativePathPattern | .....
[4] RelativePathPattern ::=
        StepPattern
        | RelativePathPattern '/' StepPattern
        | RelativePathPattern '//' StepPattern
[5] StepPattern ::=
        ChildOrAttributeAxisSpecifier NodeTest Predicate*
[6] ChildOrAttributeAxisSpecifier ::=
        AbbreviatedAxisSpecifier
        | ('child' | 'attribute') '::'
[13] AbbreviatedAxisSpecifier ::= '@'?
```

Navigationsrichtung nur zu den Blättern hin:

- Angaben `child` und `attribute` incl. Kurzformen wie üblich
- `descendant::node()` usw. ist nicht erlaubt, aber:
- `//` entspricht `descendant` !!
  (und nicht `descendant-or-self` wie sonst bei XPath!)

Knotentyptest node(): "`node()` matches any node other than an attribute node and the root node" (O-Ton XSLT-Spezifikation)
sehr tückisch, denn `node()` selektiert in XPath ausnahmslos alles!

Bedeutung eines `LocationPathPattern`:
Knoten N wird selektiert,
wenn ein Kontext K (also insb. ein Startknoten) existiert,
so daß N in der Treffermenge des LocationPathPattern beim Kontext K liegt

Alle bisherigen Beispiele sind Sonderfälle dieser allgemeinen Regel

Weitere Beispiele für `match`-Angaben:

`item[1]` oder

`item[position()=1]`
: selektiert alle `item`-Elemente, die das erste Kindelement vom Typ `item` ihres Elternelements sind

`item[last()=2]`
: selektiert alle `item`-Elemente, deren Elternelement zwei Kindelemente vom Typ `item` hat

`* [ position()=1 and self::item ]`
: selektiert alle `item`-Elemente, die das erste Kind ihres Elternelements sind

2.2 Prioritäten von Transformationsregeln

Problem: auf einen bestimmten Knoten können *mehrere Transformationsregeln anwendbar* sein, z.B.:

```
<!-- allgemeine Regel fuer alle Elementtypen -->
<xsl:template match="*"> .....</xsl:template>
....
<!-- spezielle Regel fuer einen Elementtyp -->
<xsl:template match="TelNr"> .....</xsl:template>
```

Entscheidung anhand von Prioritäten – kompliziert (diverse Möglichkeiten zur expliziten Steuerung von Prioritäten), hier nur rudimentär dargestellt

Grundregel: *speziellere Regeln (für kleinere Mengen von Knotentypen) haben höhere Priorität*

2.3 Transformationsdokumente

- ein **Transformationsdokument** enthält i.a. eine *Menge* (keine Folge) von Transformationsregeln
- die *Reihenfolge* der Transformationsregeln im Transformationsdokument ist bedeutungslos, nur Prioritäten entscheiden

→ dieser Teil von XSLT ist eine nichtprozedurale Sprache[1]

2.3.1 Kommando `xsl:include`

Gesamtmenge der Transformationsregeln kann auf mehrere Dateien verteilt werden, die dann von der Hauptdatei inkludiert werden (ermöglicht Wiederverwendung von Regeln).

Syntax:

```
<xsl:include href='uri-reference' />
```

- darf nur als Top-Level-Element (direktes Kind des `transform`-Elements) verwendet werden
- uri-reference muß korrekt sein und zu einer Datei führen
- *kein Unterschied* zwischen inkludierten und lokal definierten Regeln

2.3.2 Kommando `xsl:import`

i.w. identisch zu `xsl:include`, aber die importierten Regeln haben *geringere Priorität* als die lokal vorhandenen (im Sinne von Voreinstellungen)

Syntax:

```
<xsl:import href='uri-reference' />
```

3 Expliziter Aufruf von Transformationsregeln

3.1 Kommando `xsl:apply-templates`

Kommando `xsl:apply-templates`:

[1] man könnte auch von einer regelbasierten Sprache reden, aber das würde zu einer falschen Assoziation mit Prolog führen

- tritt in Schablonen (von Transformationsregeln oder `for-each`-Kommandos) auf
- ist ähnlich wie for-each ein Ablaufsteuerungskommando
- i.d.R. leerer Inhalt
- Parameter `select`: enthält Pfad, der die Knoten angibt, die bearbeitetet werden sollen

Implementierung von `<xsl:apply-templates select='...' />` als Pseudocode:

1. bestimme die Menge der zu bearbeitenden Knoten gemäß Parameter `select`;
   bearbeite die Knoten in der Reihenfolge gemäß Eingabe[2]
2. für jeden zu bearbeitenden Knoten *N*:
   1. bestimme den Typ von *N*
   2. bestimme die Transformationsregeln, die für Knoten dieses Typs anwendbar sind, und darunter die mit der höchsten Priorität für *N*
   3. rufe die Schablone aus dieser Transformationsregel mit *N* als Kontextknoten auf

3.2 Beispiel

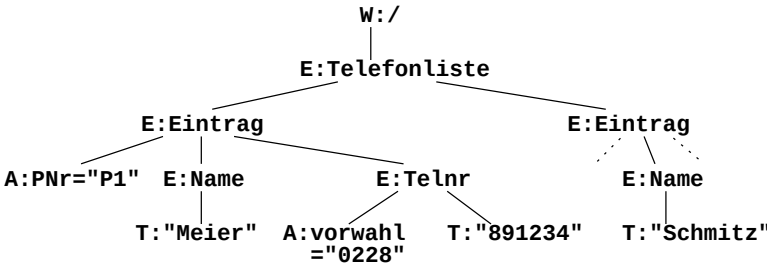
Eingabedaten:

```
<?xml version='1.0' encoding='ISO-8859-1'
        standalone='yes' ?>

<!DOCTYPE Telefonliste  [
   <!ELEMENT Telefonliste (Eintrag)* >
   <!ELEMENT Eintrag (Name, TelNr) >
   <!ATTLIST Eintrag
             PNr     ID #REQUIRED >
   <!ELEMENT Name    (#PCDATA) >
```

[2] sofern nicht zusätzliche Sortierparameter angegeben sind

```
   <!ELEMENT TelNr   (#PCDATA) >
   <!ATTLIST TelNr
             vorwahl CDATA #IMPLIED >
]>

<Telefonliste>
   <Eintrag PNr="p1" >
      <Name>Meier</Name>
      <TelNr vorwahl="0271">891234</TelNr>
   </Eintrag>
   <Eintrag PNr="p2" >
      <Name>Schmitz</Name>
      <TelNr vorwahl="0228">870887</TelNr>
   </Eintrag>
</Telefonliste>
```

zugehöriger Syntaxbaum (unvollständig):



Notation:

`W:...` Knoten, der das Wurzelelement repräsentiert
`E:...` Knoten, der ein Element repräsentiert
`T:...` Knoten, der einen Text repräsentiert
`A:...` Knoten, der ein Attribut repräsentiert

Beispiel
Aufgabe: selbst Vorwahl soll separates Unterelement werden

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<xsl:transform version='1.0'
```



```
xmlns:xsl='http://www.w3.org/1999/XSL/Transform' >

<xsl:template match='/'>
  <xsl:apply-templates  select='Telefonliste' />
</xsl:template>

<xsl:template match='Telefonliste'>
  <Telefonliste>
    <xsl:apply-templates  select='Eintrag' />
  </Telefonliste>
</xsl:template>

<xsl:template match='Eintrag' >
  <Eintrag>
    <xsl:apply-templates  select='Name' />
    <xsl:apply-templates  select='TelNr/@Vorwahl' />
    <xsl:apply-templates  select='TelNr' />
  <Eintrag>
</xsl:template>

<xsl:template match='Name'>
  <Name>
    <xsl:value-of select='.' />
  </Name>
</xsl:template>

<xsl:template match='TelNr'>
  <TelNr>
    <xsl:apply-templates  select='text()' />
  </TelNr>
</xsl:template>

<xsl:template match='@Vorwahl'>
  <Vorwahl>
    <xsl:value-of select='.' />
  </Vorwahl>
</xsl:template>

<xsl:template match='text()'>
```



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

</xsl:transform>
```

Erläuterungen / Beobachtungen:

1. Attribut `PNr` ist verlorengegangen (Attribute mit Inhalt, der von der Eingabe abhängt, können wir mit dem bisher Erlernten nicht erzeugen)

   ebenso Kommentare

2. die Elementwurzel des Ausgabebaums wird implizit erzeugt (sozusagen als Seiteneffekt der Transformationsregel mit `match='/'`), sie kann nicht explizit erzeugt werden

3. die inneren Knoten des Eingabebaums müssen i.d.R. nur kopiert werden; hierzu braucht man für jeden Elementtyp eine eigene Transformationsregel, die i.w. nach folgendem Schema aufgebaut ist:

   ```
   <xsl:template match='elementtyp'>
     <elementtyp>
       <xsl:apply-templates select='Kinder' />
     </elementtyp>
   </xsl:template>
   ```

   der Kontextknoten wird "manuell" in der Ausgabe rekonstruiert (durch Angabe des öffnenden und schließenden Tags)

   die innerhalb dieser Tags liegenden Ausgabeanweisungen erzeugen Kinder dieses Knotens

   das Kommando `<xsl:apply-templates select='Kinder' />` bearbeitet alle Kinder und ruft implizit die jeweils passende Transformationsregel auf

4. man kann mit `<xsl:apply-templates .../>` bis zu den Blättern des Eingabebaums herunterlaufen
   (Bsp.: Transformationsregeln für `TelNr` und `text()`)

   oder

bei Elementen, die nur noch Kindknoten vom Typ Text haben, schon für diese Elemente eine Transformationsregel definieren, die `<xsl:value-of select='.'/>` aufruft
(Bsp.: Transformationsregel für `Name`)

5. viel monotone Schreiberei, um die Elementstrukturen zu rekonstruieren
6. Reihenfolge der Transformationsregeln hätte beliebig anders gewählt werden können

3.3 Vorgabewert für `select`

Der Parameter `select` im Kommando `xsl:apply-templates` kann weggelassen werden;
dann gültiger Vorgabewert: `child::node()`
m.a.W.:

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

ist äquivalent zu

```
<xsl:apply-templates select='child::node()' />
```

Beispiel: in der Transformationsregel für die Telefonliste hätte man `select` weglassen können

tückisch insofern, als Attribute hiermit nicht selektiert werden!!

Testfrage [2 Minuten]: erklären Sie den Unterschied zwischen
- `<xsl:template>` und
- `<xsl:apply-templates>`

Antwort:

- `<xsl:template>` definiert eine **Transformationsregel** für einen oder mehrere Knotentypen

- `<xsl:apply-templates>` ist ein **Ausgabekommando**, das nur in einer Schablone, also z.B. im "Rumpf" einer Transformationsregel, auftritt und angibt, daß bestimmte Knotenmengen transformiert werden sollen

4 Vordefinierte Transformationsregeln

Für alle Knotentypen sind Transformationsregeln vordefiniert; diese Regeln sind sehr allgemein und haben daher geringste Priorität; übliche applikationsspezifische Regeln sind spezieller und haben höhere Priorität

Vordefinierte Transformationsregeln hängen vom Knotentyp ab:

1. für Verarbeitungsanweisungen und Kommentare:

   ```
   <xsl:template match=" processing-instruction() |
                         comment() "  />
   ```

   werden nicht ausgegeben

2. für Textknoten und Attribute:

   ```
   <xsl:template match=" text() | @* ">
     <xsl:value-of select='.' />
   </xsl:template>
   ```

   bewirkt Ausgabe des Inhalts als Text
   (*sofern* der Knoten überhaupt verarbeitet wird! bei Attributen: nicht automatisch!)

3. für Elemente und den Wurzelknoten:

   ```
   <xsl:template match=" * | / ">
     <xsl:apply-templates />
   </xsl:template>
   ```

   bewirkt im Normalfall einen rekursiven Abstieg durch den Baumstruktur der Elemente (genauer: der Kindknoten; ohne Attribute!); Verarbeitung in der Eingabereihenfolge

implizit vorhandenes voreingestelltes Hauptprogramm:
verarbeite Wurzelknoten des Syntaxbaums gemäß zug. Transformationsregel: `<xsl:apply-templates select='/' />`

Testaufgabe: Sei T ein Transformationsdokument, das keine einzige Transformationsregel enthält.

Beschreiben Sie die durch T definierte Transformation, also die Ausgabe, wenn Sie irgendeine XML-Datei mit T transformieren.

5 Die identische Transformation und Projektionen

Die folgende Transformationsregel reproduziert den Eingabesyntaxbaum unverändert:

```
<xsl:template match=" node() | @* " >
  <xsl:copy>
    <xsl:apply-templates select=" node() | @* " />
  </xsl:copy>
</xsl:template>
```

Erläuterungen:

- `node()` selektiert alle Knotentypen außer Attributen und dem Wurzelknoten
- `@*` selektiert alle Attribute
- Wurzelknoten wird speziell behandelt - hier kein Thema
- `xsl:copy` ist ein Kommando, das den Kontextknoten in den Ausgabesyntaxbaum kopiert (hat keinen `select`-Parameter)

  bildlich gesprochen: erzeugt vor allem die *tags*

  der Inhalt ist eine Schablone – diese wird ausgeführt und alle erzeugten Knoten werden an den kopierten Knoten als Kind angehängt

Nutzung für "Projektionen". "Entwurfsmuster":

- identische Transformation als allgemeinste Regel definieren (lokal in der gleichen Datei oder in einem imporierten oder inkludierten Transformationsdokument)
- für auszublendende Attribute oder Elemente (incl. darunterliegende Teilbäume!) Transformationsregeln mit leerer Schablone definieren

Beispiel: Lösche die `TelNr`-Elemente in der Telefonliste

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<xsl:transform version='1.0'
  xmlns:xsl='http://www.w3.org/1999/XSL/Transform' >

  <xsl:template match=" node() | @* " >
    <xsl:copy>
      <xsl:apply-templates select=" node() | @*  " />
    </xsl:copy>
  </xsl:template>

  <xsl:template match='TelNr' />

</xsl:transform>
```

Literatur

[XPAT] Kelter, U.: Lehrmodul "XPATH"; 2007

[XSLT] Kelter, U.: Lehrmodul "XSLT, Teil 1 (Stichworte)"; 2007

