XSLT, Teil 1 (Stichworte)

Udo Kelter

02.06.2009

Zusammenfassung dieses Lehrmoduls

Die Transformationssprache XSLT (eXtensible Stylesheet Language for Transformation) ist ein Teil des XSL-Standards. In der XSLT-Sprache können Transformatoren definiert werden, die eingegebene XML-Dateien in neue XML-Dateien, aber auch andere Formate transformieren. Dieses Lehrmodul stellt die Grundkonzepte der XSLT vor, ferner die grundlegenden Ausgabe- bzw. Steuerkommandos value-of und for-each.

Vorausgesetzte Lehrmodule:

obligatorisch:
- Transportdateien und die XML

empfohlen:
- Transportdateien und die SGML
- XML-Namensräume

Stoffumfang in Vorlesungsdoppelstunden: 1.0

Inhaltsverzeichnis

1 Exkurs: Transformation von XML-Dokumenten

Abfragesprachen für XML-Datenbestände (XPath, XQuery) weisen einige prinzipielle Unterschiede zu Abfragesprachen für relationale DB auf:

- Abfragen werden als Dokumenttransformationen verstanden
  Input einer Abfrage: XML-Datei
  Output: normalerweise wieder eine XML-Datei!
- Abfragesprache wird durch technische Ausführungsumgebung geprägt – bei XML: WWW-Technologien

1.1 Transformation von konzeptuellen Daten in externe Darstellungen

DB-Inhalte / XML-Daten sind nur Inhalte, keine Darstellung → müssen (ggf. nach Selektion, Sortierung, ...) in darstellbare Form *transformiert* werden:

1. Hinzufügen von Überschriften, Feldnamen, usw.
2. Festlegung von Fonts, Schriftgrößen, Farben, Hintergründen, sonstigen *Stil*elementen [ggf. vom Benutzer gesteuert!!]

Vorgehen bei relationalen Abfragesprachen:
Abfrageergebnisse (Tabellen) sind in Rohform generell ungeeignet für Anzeige
→ ausgeprägte **Arbeitsteilung zwischen Query-Prozessor und Applikation**:

- DBMS zuständig für:
  - Selektion bestimmter Daten
  - Sortierungen

  - Gruppierungen, Aggregation
  - Verbinden separater Datenbestände
- Applikation zuständig für:
  - Layoutgenerierung, Algorithmen
  - ggf. Aufsammeln von Daten aus mehreren Relationen

Generierung der externen Darstellung für WWW-Schnittstellen:

- über Seitenbeschreibung in HTML (nicht über Graphik-API), also *Transformation von XML nach HTML*
- HTML strukturell ähnlich wie XML

  → gleiche Technologien für Abfragedienste und Layoutaufbereitung, oft **keine klare Trennung zwischen beiden Funktionen**
- Layoutaufbereitung serverseitig und clientseitig möglich
- zusätzliche Konfusion durch clientseitige, an HTML gebundene Technologien zur Layoutaufbereitung, insb. Cascading Style Sheets (CSS)

→ sehr viele Architekturvarianten

1.2 Abfragen als Dokumenttransformationen

Grundgedanke:

- Eingabedaten = XML-Datenbank = 1 großer abstrakter Syntaxbaum (oder mehrere Bäume)
- Ausgabedaten: sollten auch einen Syntaxbaum bilden!!
  (u.a. wegen Hintereinanderschaltung von Abfragen)

somit:

- 1 Abfrage erzeugt Knoten verschiedenen Typs (nicht nur eines Typs wie in rel.DB)

- Abfrage erzeugt auch Baumstruktur
- Abfrage kann eher als **Transformation des Eingabebaums in einem Ausgabebaum** angesehen werden

1.3 Funktionsmerkmale von XML-Abfragesprachen

XML-Abfragesprachen + erforderliche ergänzende Technologien sind komplizierter als relationale Abfragesprachen – wieso?

über die reinen Abfragedienste hinaus:
erforderliche Funktionsmerkmale für den Aufbau eines auszugebenden Syntaxbaums:

1. Funktionen zum *Erzeugen von Elementknoten* (nur Baumstruktur); oft einfache Kopien von Knoten des Eingabebaums
   ggf. völlig neue Knotentypen
2. Funktionen zum *Berechnen des Inhalts von Elementen*
   trivial: ausgeben von atomaren Felderinhalten
   kompliziert (in analoger Form nicht in rel.DB vorhanden): verarbeiten und erzeugen von `ANY`- und `MIXED`-Inhalten
   endet schnell bei allgemeinen Textverarbeitungsproblemen; reine Datenextraktion und Weiterverarbeitung der Daten oft nicht trennbar
   zugehörige Teile der XML-Technologien (Abfragesprache + ergänzendes) entsprechen einer allgemeinen Programmiersprache + Textprozessor
3. Funktionen zum *Erzeugen von Attributen*
4. Funktionen zum *Berechnen des Inhalts von Attributen* (andere Restriktionen als bei Elementinhalten)
5. Funktionen für "low level"-Probleme (physische Ebene; Zeichensätze usw.)

6. gleichzeitiges Erzeugen verschiedener Knotentypen
   der *komplette Ausgabebaum muß in 1 Verarbeitungsschritt aufgebaut werden*
   nach Typen getrenntes Aufsammeln einzelner Knoten durch die Applikation (analog zu mehreren SQL-Abfragen) ist nicht sinnvoll / nicht machbar, dito Imitieren von relationalen Verbunden

2 Einordnung von XSLT

Bestandteile der **Extensible Stylesheet Language** (**XSL**):

- Transformationssprache **XSLT** (**eXtensible Stylesheet Language for Transformation**)
  XSLT definiert i.w. Transformationsregeln und Ausgabeanweisungen – Details s.u.
  XSLT basiert auf **XPath**
- Formatierungssprache: Extensible Stylesheet Language (**XSL**) Version 1.1, W3C Recommendation, 05 December 2006, `http://www.w3.org/TR/xsl11/`
  – hier nicht betrachtet

insg. ziemlich komplexe Technologie

XPath:

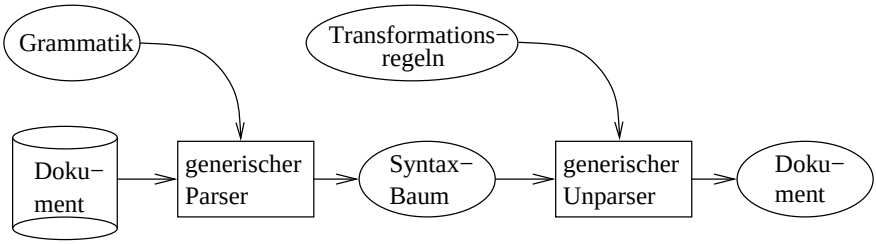
- definiert Pfadausdrücke;
  Pfadausdruck: bestimmt ausgehend von einem Startknoten eine Menge von Zielknoten im Eingabesyntaxbaum
- beinhaltet *keine Ausgabeanweisungen*

Versionen von XSLT / XPath:

- XSLT 1.0, von 1999, basiert auf XPath 1.0
- XSLT 2.0, verabschiedet 2007, basiert auf XPath 2.0
  ist wesentlich ausgereifter

3 XSLT-Prozessoren

3.1 Grundlegende Funktionsweise von XSLT-Prozessoren



Schritte:

1. gegebene XML-Datei(en) in einen Syntaxbaum übersetzen
2. den Baum durchlaufen (meistens ähnlich preorder);
   bei jedem besuchten Knoten eine Transformationsregel anwenden

Transformationsregeln können bunt gemischt Abfragedienste und Layoutaufbereitung betreffen!

Varianten der Ausgabe:

1. Syntaxbaum, in XML dargestellt
2. Seiteninhalt, in HTML dargestellt
3. beliebiger Text (z.B. in LaTeX-Syntax für Druckausgabe)

Varianten der Eingabeverarbeitung:
- validierend (ggf. optional) oder nicht validierend

technische Einbettung:

- nichtinteraktive selbständige Programme;
  Beispiele: `xsltproc`, `Xalan`, `Saxon`, ...
  s. WWW-Seite für die Übungen

- nichtinteraktive dynamisch gebundene Bibliotheken; Beispiel: `libxslt`; als Teil von
  - WWW-/Applikationsservern (ggf. inkl. Weiterverarbeitung zu HTML)
  - WWW-Browsern (inkl. sofortiger Weiterverarbeitung in HTML und in die Bildschirmdarstellung – unsicher, nicht in allen Browsern vorhanden / kompatibel)

3.2 Eingabe

zulässige Eingabe:
- nur wohlgeformtes und ggf. gültiges XML, wohlgeformtes HTML
- *kein* nicht wohlgeformte HTML-Fragmente, kein PDF, kein ...

Arten von Knoten des eingelesenen Syntaxbaums aus Sicht von XSLT:
- Dokumentwurzel
- Elemente
- Textabschnitte (1 TA = Text zwischen zwei aufeinanderfolgenden *tags*)
- Attribute
- Namensräume
- Processing Instructions
- Kommentare

kein Bestandteil der Eingabe: die zug. Dokumenttypdefinition, wenn vorhanden [allenfalls indirekt wegen Vorgabewerten]

Dokumentwurzel:
- entsteht implizit beim Einlesen;
  ist *nicht* durch ein Element in der Eingabedatei repräsentiert
- hat bei einer wohlgeformten Datei genau einen Element-Kindknoten mit Typ, der in der DOCTYPE-Klausel angegeben ist, andernfalls i.a. mehrere Kindknoten

3.3 Ausgabe

Ausgabe im Normalfall wieder eine XML-Datei!

- ist wohlgeformt
  → sollte als textuelle Darstellung eines (Ausgabe-) Syntaxbaums angesehen werden!!
- kann erneut transformiert werden
- inkl. XML-Deklaration (`<?xml version='1.0'...?>`)
- *ohne* DTD – im Ausgabesyntaxbaum können Teile fehlen / Knoten umsortiert sein / Strukturen verändert sein / ...
  passende DTD i.a. nicht automatisch bestimmbar

Auswahl der Ausgabemethode:
XML-Ausgabe ungeeignet für HTML-Prozessoren, LaTeX usw. (wegen `<?xml ... ?>` Anweisung in 1. Zeile, unpassender Zeichensatzcodierung, ...)
→ Die meisten XSLT-Prozessoren unterstützen mehrere Ausgabemethoden; Auswahl mit der Steueranweisung `xsl:output` und Parameter `method`; zulässige Werte:

`xml`	Vorgabe
`html`	Auswahl mit `<xsl:output method='html' />`
`text`	Auswahl mit `<xsl:output method='text' />`

muß auf oberster Ebene im Transformationsdokument stehen,
am besten direkt hinter dem öffnenden `xsl:transform`-*tag* (s.u.)

weitere Steuerungsangaben als Attribute von `xsl:output`:

`version` zur Zeit nur 1.0 erlaubt

`omit-xml-declaration`
erlaubte Werte: `yes` und `no`. Angabe `yes`: keine XML-Deklaration in der Ausgabe

`standalone` erlaubte Werte: `yes` und `no`. Wert wird in die ausgegebene XML-Deklaration übernommen

`encoding` erlaubte Werte: alle vom Prozessor unterstützten Zeichensatzcodierungen, z.B. ISO-8859-1, UTF-8, UTF-16, ...; Vorgabewert: UTF-8

`indent` erlaubte Werte: `yes` und `no`. Angabe `yes`: Prozessor kann versuchen, in der Ausgabe Leerraum so zu verändern, daß gut lesbare Einrückungen entstehen

4 XSLT-Transformationsdokumente

XSLT-Transformationsdokumente:

- enthalten i.w. Transformationsregeln
- synonym: **XSLT-Stylesheet-Dokumente**, **XSLT-Stylesheet**[1]

4.1 Äußerer Aufbau eines Transformationsdokuments

Muster:

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

<xsl:transform version='1.0'
   xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:output ....  />
...
<xsl:template ...> ....  </xsl:template>
<xsl:template ...> ....  </xsl:template>
...
</xsl:transform>
```

[1] nicht verwechseln mit Cascading Stylesheets! Deren Zweck ist teilweise ähnlich zu XSLT-Stylesheets, sind aber Teil der Begriffswelt von HTML; steuern das rendering bei der Interpretation von HTML-Text

- ist ein XML-Dokument! (→ gut lesbar ???!?)
- muß den Namensraum
  `http://www.w3.org/1999/XSL/Transform`
  für die Notation der Transformationsregeln nutzen
  dieser wird i.d.R. auf den Namensraumbezeichner `xsl` abgebildet
- Wurzelelementtyp eines Transformationsdokuments:
  `stylesheet` oder `transform` (sind äquivalent)
- zulässige Kindelementtypen des `transform`-Elements i.w.:
  - `output`-Elemente: Ausgabesteuerung; müssen zuerst kommen
  - `template`-Elemente: Transformationsregeln

4.2 Syntax von XSLT-Kommandos

- Aufschreibung als XML-Element
- *Kommandoname* = Elementtypname
  Beispiel: `<xsl:template ...> .... </xsl:template>`
  oft leerer Inhalt → kompakte Notation sinnvoll
- *Parameter*: i.d.R als Attribute des Elements
  Beispiel: `<xsl:output method='text' />`

linguistische Qualität: vergleichbar mit Assembler `>:-[`

4.3 Transformationsregeln – Crashkurs

Funktionsweise der Transformationsregeln ist *ziemlich* kompliziert

daher: zunächst weitgehend vermeiden (nicht ganz stilgerecht) und zunächst anderen Stoff erlernen

Vermeidungsstrategie: nur mit einer Transformationsregel für die Dokumentwurzel arbeiten

Konzeptueller Inhalt einer Transformationsregel (einfachste Form):

1. *Knotentyp*, auf den die Regel anzuwenden ist
2. *Schablone* / "Textmuster" (*template*), das für einen Knoten dieses Typs auszugeben ist;
   enthält u.a. Verarbeitungsanweisungen

Notation einer Transformationsregel: in Form eines `xsl:template`-Elements (kryptisch):

```
<xsl:template match='Knotentyp' >
      .... Schablone ....
</xsl:template>
```

- *Knotentyp* im Parameter `match`
- *Schablone* im Inhalt des `template`-Elements

5 Arbeiten mit einer einzigen Transformationsregel

- nur 1 Transformationsregel, gilt für Dokumentwurzel
- eigentlich nicht ganz stilgerecht
- leichterer Einstieg

Transformationsregel für die Dokumentwurzel
Knotentyp = `/`; Spezifikation:

```
<xsl:template match='/'>
      .... Schablone ....
</xsl:template>
```

Wird zu Beginn der Verarbeitung automatisch ausgeführt!

5.1 Aufruf einer Schablone

- explizite Aufrufe sind möglich – Details später
- Aufrufe sind sehr häufig implizit (kein sichtbarer Quelltext)
  z.B. Schablone der Transformationsregel für die Dokumentwurzel
- bei jedem Aufruf wird ein **Kontext** erzeugt, in dem die Schablone ausgeführt wird;

Bestandteile des Kontextes eines Schablonenaufrufs:

1. **Kontextknoten**: aktuell zu verarbeitender Knoten
   Beispiel: Dokumentwurzel
   hat Typ gemäß `match`-Parameter
2. Numerierungen u.a., vorerst uninteressant

5.2 "Ausführen" einer Schablone

Schablone = Inhalt des `template`-Elements
= Folge von Textknoten, Elementknoten, Kommentaren...
Verarbeitungsregeln:

1. gib in der Schablone enthaltene Textknoten wörtlich aus;
   Ausnahme: Textknoten, die nur Leerraum enthalten, werden ignoriert
2. gib die in der Schablone enthaltenen Element- und Attributknoten "wörtlich" aus, Ausnahme:
3. wenn ein Elementknoten eine Ausgabe- oder Steueranweisung darstellt: führe diese Anweisung aus

Einstiegsbeispiel:


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

<xsl:template match='/'>
   <b x='3'>Hello World!</b>
</xsl:template>

</xsl:transform>
```

5.3 Ausgabe- bzw. Steueranweisungen

- sind alle Elemente, deren Typ im reservierten Namensraum für Anweisungen liegt

  normalerweise: der Tag-Name fängt mit `xsl:` an
  (Details s. Lehrmodul über Namensräume)

  Beispiel: `<xsl:value-of .... />`
- haben einen Parameter *select*, in dem i.a. ein relativer Pfad steht, der den oder die zu bearbeitenden Knoten angibt

  Beispiel: `<xsl:value-of select='`*pfad*`' />`

  – von wo aus relativ navigieren???
- haben alle einen *versteckten Parameter*: den Kontextknoten der Schablonenausführung – wird als Ausgangsknoten für die relativen Pfade benutzt

5.4 Das Kommando `xsl:value-of`

Wandelt einen beliebigen Eingabeknoten in einen ausgegebenen Textknoten um

- stets leerer Inhalt
- hat einen Parameter `select`, der einen Knoten auswählt

Wirkung: erzeugt einen Textknoten im Ausgabebaum, ausgegebener Text ist:

- bei Textknoten: der Text
- bei Attributen: der Attributwert – sehr häufige Verwendung!
- bei Elementen: Konkatenation aller Texte in allen Kind- und Nachfahre-Knoten des Teilbaums

Beispiel:

- Eingabedaten: eine Lehrmodulbeschreibung
- Ausgabe: *Lehrmodulname* (*Kürzel*) – *Verantwortlicher*

Einfache Lösung:

```
<xsl:template match='/'>
  <xsl:value-of select='//LEHRVERANSTALTUNGSNAME' />
  (<xsl:value-of
select='//LEHRVERANSTALTUNGSKUERZEL'/>)
  --
  <xsl:value-of
select='//VERANTWORTLICHER/@dozentId' />
</xsl:template> .....
```

u.U. störend: viel Leerraum in der Ausgabe

5.5 Das Kommando `xsl:text`

Viel Trickserei in XSLT bei der Behandlung von Leerraum im Eingebebaum und in den Schablonen (s. Abschnitt "Whitespace Stripping")

simple Methode, um Leerraum zu vermeiden, der in den Schablonen generiert wird:

Kommando `xsl:text`: gibt den enthaltenen Text aus

Funktion `normalize-space(..)`: entfernt Leerraum vorne und hinten und reduziert in der Mitte auf 1 Leerzeichen

obiges Beispiel, bessere Lösung:

```
....
<xsl:template match='/'>
  <xsl:value-of select=
    'normalize-space(//LEHRVERANSTALTUNGSNAME)' />
  <xsl:text> (</xsl:text>
  <xsl:value-of select='//LEHRVERANSTALTUNGSKUERZEL'
/>
  <xsl:text>) -- </xsl:text>
  <xsl:value-of
select='//VERANTWORTLICHER/@dozentId' />
  <xsl:text>
</xsl:text>
</xsl:template> ....
```

5.6 Das Kommando `xsl:for-each`

Merkmale des Ausgabekommandos `xsl:for-each`

1. hat einen Parameter `select`, der i.d.R. einen XPath-Ausdruck enthält

   = *Menge der Knoten des Eingabebaums, über die iteriert werden soll*

   (meistens direkte Kindknoten)
2. Inhalt: eine "innere" Schablone, aufgebaut analog zum Inhalt einer Transformationsregel
3. zusätzlich Sortierung der Knotenmenge möglich

Wirkung:

- für jeden durch das `select` gewählten Knoten wird die innere Schablone ausgeführt
- dieser Knoten ist der *Kontextknoten* für diese Ausführung der *inneren Schablone*!

  → Wechsel des Kontextknotens beim Betreten der inneren Schablone!

Beispiel: wie oben, aber zusätzlich alle Semester (mit Komma getrennt) angeben, in denen das Lehrmodul angeboten wurde

1. Lösung:

```
<xsl:template match='/'>
  <xsl:value-of select='//LEHRVERANSTALTUNGSNAME' />
  (<xsl:value-of select='//LEHRVERANSTALTUNGSKUERZEL' />) --
  <xsl:value-of select='//VERANTWORTLICHER/@dozentId' />
  <xsl:text> </xsl:text>
  <xsl:for-each select="//DURCHFUEHRUNG" >
     <xsl:value-of select='./@semester' />,
  </xsl:for-each>
</xsl:template>
```

nicht ganz perfekt: Komma am Ende zuviel

2. Lösung:

```
<xsl:template match='/'>
  <xsl:value-of select='//LEHRVERANSTALTUNGSNAME' />
  (<xsl:value-of select='//LEHRVERANSTALTUNGSKUERZEL' />) --
  <xsl:value-of select='//VERANTWORTLICHER/@dozentId' />
  <xsl:text> </xsl:text>

  <xsl:for-each select="//DURCHFUEHRUNG[ last()>position() ]">
     <xsl:value-of select='./@semester' />,
  </xsl:for-each>

  <xsl:value-of select='//DURCHFUEHRUNG[ last() = position() ]
     / @semester' />
</xsl:template>
```

Literatur

[XPath99] XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999; http://www.w3.org/TR/xpath; 1999

[XSLT99] XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999; http://www.w3.org/TR/xslt;

1999

[XML] Kelter, U.: Lehrmodul “XML”; 2004

