

XML Vorlesung ETHZ, Sommersemester 2006

XSL Transformations (XSLT) Teil III

Erik Wilde

13.6.2006

<http://dret.net/lectures/xml-ss06/>

13.6.2006

XML Vorlesung ETHZ SS 2006

1

Übersicht

- Tips und Tricks
 - Mengenoperationen in XPath
- Keys
- Modularisierung
 - Import Precedence
- XSLT Extensions
- XSLT und Schemas
 - Schemas für die Eingabe
 - Schemas für die Ausgabe

13.6.2006

XML Vorlesung ETHZ SS 2006

2

Identitäts-Test mit generate-id()

- Test auf Node Identität oft notwendig
 - z.B. Node in Variable gespeichert
 - Test auf Identität mit Variable in einer Schleife
- der direkte Vergleich ist nicht korrekt
 - aber: $\$x = \y ist ein korrekter XPath
 - er vergleicht jedoch die String Values der Knoten
 - oft richtig, aber nicht immer (schwer zu findender Fehler)
- Nodes müssen über ihre ID verglichen werden
 - $\text{generate-id}(\$x) = \text{generate-id}(\$y)$
 - vergleicht die generierten IDs
 - diese müssen identisch sein bei identischen Nodes

13.6.2006

XML Vorlesung ETHZ SS 2006

3

Mengenoperationen

- XSLT kennt nur den Union Operator (|)
 - die am häufigsten gebrauchte Operation
 - aber nicht die einzig sinnvolle Operation
- Ausnutzung der Definition von XPath Prädikaten
- Verwendung von Schnittmengen
 - $\$a[\text{count}(. | \$b) = \text{count}(\$b)]$
 - selektiert Nodes, die in \$a und in \$b sind
- Verwendung von Differenzmengen
 - $\$a[\text{count}(. | \$b) \neq \text{count}(\$b)]$
 - selektiert Nodes in \$a, die nicht in \$b sind

13.6.2006

XML Vorlesung ETHZ SS 2006

4

Zugriff auf Nodes über Keys

- Deklaration mit `<xsl:key>`
 - immer auf dem Top Level deklariert
 - Name, Match Pattern und Wert des Keys
 - falls Wert ein Node Set ergeben sich mehrere Keys
 - `<xsl:key name="" match="name" use="vorname"/>`
 - Hinweis an den XSLT Prozessor
 - sinnvollerweise wird ein Index aufgebaut
- Keys müssen nicht eindeutig sein
 - Eindeutigkeit muss separat sichergestellt werden
 - in XSLT: `count(key(. . .))` darf höchstens eins sein
 - im Schema: DTD I Ds oder XML Schema Identity Constraints

13.6.2006

XML Vorlesung ETHZ SS 2006

5

Deklarieren und Verwenden eines Key

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:key name="topici d" match="topi c" use="@TID"/>

  <xsl:template match="/">
    <xsl:value-of select="key('topici d', 'asci i')/name"/>
    <xsl:text> (</xsl:text>
    <xsl:value-of select="key('topici d', 'asci i')/ali as"/>
    <xsl:text>), </xsl:text>
    <xsl:value-of select="key('topici d', 'md5')/name"/>
    <xsl:text> (</xsl:text>
    <xsl:value-of select="key('topici d', 'md5')/ali as"/>
    <xsl:text>)</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

13.6.2006

XML Vorlesung ETHZ SS 2006

6

Multivalued Keys

- `<xsl:key>` ist sehr flexibel
 - ein Key kann mehrere Werte für einen Node haben
 - `<xsl:key name="" match="name" use="vorname"/>`
- wie findet man Schnittmengen von Keys?
 - generelles Problem in XSLT/XPath
 - auf Mengen gibt es nur den Union Operator (|)
- spezieller Code in XSLT


```
<xsl:variable name="a" select="key(...)" />
<xsl:variable name="b" select="key(...)" />
<xsl:variable name="result"
  select="$a[count(.|$b) = count($b)]"/>
```

13.6.2006

XML Vorlesung ETHZ SS 2006

7

Identifizieren von Knoten

- `generate-id()` erzeugt eine eindeutige ID
 - eine gültiger XML Name (DTD ID/IDREF)
 - kann jedem Knoten einen Namen zuweisen
- keine inverse Funktion
 - Abbildung ID auf Knoten nicht unterstützt
 - möglich durch `//*[generate-id()=$ID]`
 - extrem ineffizient
- Schlüsseldefinition für inverses Mapping
 - `<xsl:key name="id" match="*" use="generate-id()" />`
 - Benutzung einfach mit `key('id', $ID)`

13.6.2006

XML Vorlesung ETHZ SS 2006

8

Modularisierung

- lange und unübersichtliche Style Sheets
 - sowieso problematisch wegen der XML-Syntax
 - Trennung in verschiedene Teile
- Verwendung mehrerer Teile
 - bessere Übersichtlichkeit
 - Wiederverwendung von Style Sheets
 - Ergänzung von Teilen in verschiedenen Kontexten
 - Wiederverwendung von Template Rules

Varianten der Modularisierung

- textuelles Einbinden
 - `<xsl : i ncl ude>` als Top Level Element
- logisches Importieren
 - `<xsl : i mport>` für die logische Strukturierung
 - importierte Teile haben niedrigere Priorität
 - `<xsl : appl y-i mports>` für die Ausführung importierter Template Rules

Struktur eines XSLT Programms

- ein Style Sheet ist die Wurzel
 - als Style Sheet dem XSLT Prozessor übergeben
 - als Style Sheet im XML Dokument angegeben
- weitere Teile werden referenziert
 - `<xsl : i ncl ude>` für textuelles Einbinden
 - `<xsl : i mport >` für logisches Importieren
 - beliebig weit rekursiv fortgesetzt
 - Resultat ist ein Baum von Style Sheets
- Zusammensetzung vor der Ausführung
 - alle referenzierten Teile müssen existieren

13.6.2006

XML Vorlesung ETHZ SS 2006

11

Textuelles Einbinden von Stylesheets

- `<xsl : i ncl ude>` für textuelles Einbinden
 - muss auf dem Top Level erscheinen
 - entspricht dem Copy&Paste des Style Sheets
 - abgesehen von dessen Document Element
- nicht tauglich für lose gekoppelte Style Sheets
 - vergleichbar einem simplen Preprocessor
 - keinerlei Einfluss auf die Interpretation
- lose gekoppelte Style Sheets
 - logisches Einbinden mit niedrigerer Priorität
 - hat einen erwünschten Einfluss auf die Interpretation
 - definierte Dinge können überschrieben werden

13.6.2006

XML Vorlesung ETHZ SS 2006

12

Importieren von Stylesheets

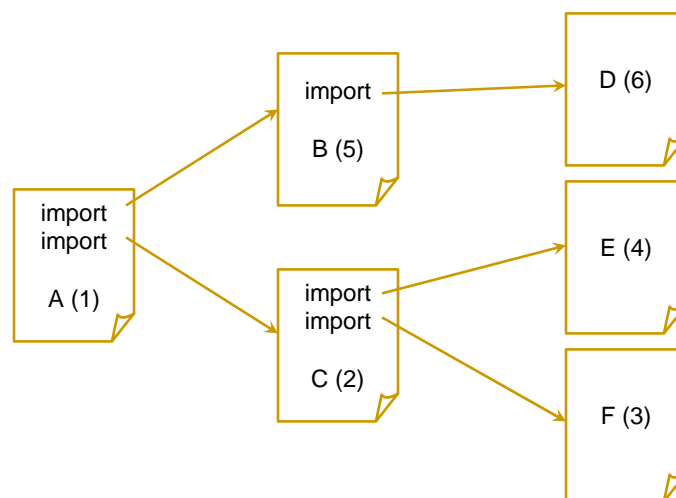
- `<xsl:import>` für logisches Importieren
 - muss auf dem Top Level erscheinen
 - muss als erstes Element auf Top Level erscheinen
- Imports definieren die *Import Precedence*
 - Importiertes geringer gewichtet als das Importierende
 - zuerst importiert ist geringer gewichtet
- Includes ändern die Import Precedence nicht
 - gleich gewichtet wie das includende Style Sheet

13.6.2006

XML Vorlesung ETHZ SS 2006

13

Import Precedence am Beispiel



13.6.2006

XML Vorlesung ETHZ SS 2006

14

Ergänzen von Template Rules

- Redefinieren importierter Template Rules
 - von XSLT ausdrücklich erlaubt
 - wird durch die *Import Precedence* geregelt
- u.U. soll die Template Rule ergänzt werden
 - neuer Code im eigenen Style Sheet
 - bestehender Code im importierten Style Sheet
- `<xsl : apply- imports >` in Template Rule
 - suche nach einer passenden Template Rule
 - definiert anhand der *Current Template Rule*
 - gleicher Mode
 - in einem importierten Style Sheet

13.6.2006

XML Vorlesung ETHZ SS 2006

15

Zusammenfassung Modularisierung

- XSLT bietet Modularisierung
 - in zwei unterschiedlichen Varianten
- XSLT ist tendenziell unübersichtlich
 - Grund dafür ist die XML Syntax
 - eher kürzere Programme schreiben
- `include` ermöglicht textuelles Kombinieren
- `import` ermöglicht logisches Kombinieren
 - die *Import Precedence* definiert eine Hierarchie

13.6.2006

XML Vorlesung ETHZ SS 2006

16

XSLT Extensions

- XSLT ist eine einfache Sprache
 - Design für die Ausführung in einfacher Umgebung
 - Design für die Ausführung einfacher Programme
- XSLT wird für vieles andere eingesetzt
 - oft Server-seitig in B2B Anwendungen
 - wesentlich weitergehende Anforderungen
- viele XSLT Prozessoren bieten Erweiterungen
 - praktisch und viele Erleichterungen
 - Bindung an einen spezifischen XSLT Prozessor
 - Sammlung als EXSLT (<http://www.exslt.org>)

13.6.2006

XML Vorlesung ETHZ SS 2006

17

Umgang mit Extensions

- Extensions werden mit Namespaces identifiziert
 - eindeutige Identifikation
 - Extensions werden mit QNames angesprochen
 - gilt für Elemente, Attribute und Funktionen
- Abfrage von vorhandenen Extensions
 - `element-available()` Funktion
 - `function-available()` Funktion
- definierter Fallback-Mechanismus
 - `<xsl:fallback>` innerhalb eines Elements
 - wird nur im Fehlerfall ausgeführt
 - kann Ersatzfunktionalität oder Terminierung veranlassen

13.6.2006

XML Vorlesung ETHZ SS 2006

18

Extension Elemente und Attribute

- definieren zusätzliche Anweisungen
 - benutzen QNames aus einem Extension Namespace
 - sind spezifisch für einen XSLT Prozessor
- Saxon definiert
 - `<saxon: whi l e>` für konditionale Iterationen
 - `<saxon: assi gn>` für Wertzuweisung zu Variablen
 - Variable muss auf `saxon: assi gnabl e="yes"` gesetzt sein
- Xalan definiert
 - `<xal an: wri te>` für mehrere Ausgabedokumente
 - `<xal an: for-each-token>` für Token-Schleifen
- weitere Prozessor-spezifische Erweiterungen

13.6.2006

XML Vorlesung ETHZ SS 2006

19

Extension Functions

- definieren zusätzliche Funktionen
 - benutzen QNames aus einem Extension Namespace
 - erweitern die XSLT/XPath Function Library
- Saxon definiert
 - `eval uate` für dynamische XPath Expressions
 - `path generiert` XPath für den Context Node
- Xalan definiert
 - `i ntersect i on` für Node Set Schnittmengen
 - `I i ne-number` für Context Node Zeilennummer
- weitere Prozessor-spezifische Erweiterungen

13.6.2006

XML Vorlesung ETHZ SS 2006

20

Extensions selbstgemacht

- Integration eigener Funktionen in XSLT
 - in Form von eigenen Elementen
 - in Form von eigenen Attributen
- spezifisch für einen XSLT Prozessor
 - kein etablierter Standard für ein API
 - Implementierung ist nicht (einfach) portabel
- Implementierung eigener Elemente
 - kompliziertere Integration (mehrere Phasen)
- Implementierung eigener Funktionen
 - erhalten Context und Argumente übergeben

13.6.2006

XML Vorlesung ETHZ SS 2006

21

XSLT und Schemas für XML

- XSLT verarbeitet well-formed XML
 - Eingabe ist ein Infoset (oder DOM)
 - Verarbeitung gemäss der Regeln des Programms
- es gibt verschiedene Schemasprachen für XML
 - DTDs aus dem XML Standard selber
 - XML Schema als Weiterentwicklung von DTDs
 - separater Standard, zunehmende Akzeptanz
 - Schematron als regelbasierte Schemasprache
 - im Gegensatz zu den Grammatiken von DTD & XML Schema
 - in vielen Anwendungsfällen ist valides XML nötig
 - B2B Anwendungen erwarten korrekte Daten

13.6.2006

XML Vorlesung ETHZ SS 2006

22

Warum Schemas für die Eingabe?

- XSLT Code basiert auf Annahmen
 - die XPath Ausdrücke
 - /personen/person/name/nachname vs. //nachname
 - die Struktur der Templates
- invalides XML verletzt diese Annahmen
 - Fehler in der Verarbeitung
 - unvorhersehbare Resultate
- Annahmen sind wichtig beim Programmieren
 - ohne Schema programmiert es sich mühsam
 - Kenntnis des Schemas ermöglicht Effizienz

13.6.2006

XML Vorlesung ETHZ SS 2006

23

Schemas für die XSLT Eingabe

- Parser liefert dem XSLT Prozessor das DOM
 - manchmal austauschbar, manchmal eingebaut
- Parser kann parametrisiert werden
 - Validierung als Schritt vor dem XSLT Prozessor
 - DTD und/oder XML Schema Validierung
 - falls der Parser auch XML Schema unterstützt
 - kann per Parser Option eingestellt werden
- invalides XML bricht die Verarbeitung ab
 - Eingabebedingung nicht erfüllt
 - Korrektur oder Rückweisung (je nach Szenario)

13.6.2006

XML Vorlesung ETHZ SS 2006

24

Warum Schemas für die Ausgabe?

- oftmals wird das Resultat weiterverarbeitet
 - Eingabe in eine weitere Verarbeitungsstufe
 - die ihrerseits auf gewissen Annahmen basiert
 - Ausgabe sollte Regeln gehorchen
- Ausgabe wird dynamisch zusammengesetzt
 - abhängig vom Aufbau des Eingabedokuments
- Struktur der Ausgabe schwer vorhersehbar
 - einfacher bei *Pull-Processing* Style Sheets
 - schwieriger bei *Push-Processing* Style Sheets
 - immer schwierig bei nicht-trivialen Style Sheets

13.6.2006

XML Vorlesung ETHZ SS 2006

25

Schemas für die Ausgabe

- XSLT erzeugt einen Node Tree
 - dieser wird als well-formed XML serialisiert
 - u.U. wird er optimiert (z.B. Namespace Nodes)
 - u.U. wird er direkt als Baum (DOM) weitergegeben
- Validierung des Resultats ist einfach
 - Parser verwenden und validieren lassen
 - weiterer Schritt in der Verarbeitungspipeline
- Validierungen sind u.U. aufwendig
 - stark abhängig von Schema und Parser
 - nur an strategischen Stellen einsetzen

13.6.2006

XML Vorlesung ETHZ SS 2006

26

XSLT 2.0 vs. XSLT 1.0

- starke Erweiterung des Sprachumfangs
- Result Tree Fragments gestrichen
 - jeder XPath-Ausdruck gibt eine Sequence zurück
- auf Sequences kann mit XPath operiert werden
- Unterstützung der XML Schema Datentypen
- Sequences können eine Mischung von Objekten dieser Datentypen enthalten

13.6.2006

XML Vorlesung ETHZ SS 2006

27

Wichtige Erweiterungen

- Sortieren und Gruppieren
- benutzerdefinierte Funktionen
- Zeichenkettenverarbeitung
 - Unterstützung von regulären Ausdrücken
- Schreiben mehrerer Ausgabedokumente und Formate in einem Durchgang

13.6.2006

XML Vorlesung ETHZ SS 2006

28

Sortieren und Gruppieren

- neues Top Level Elemente xsl:sort-key
- xsl : sort -key definiert einen benannten Sortierschlüssel
- xsl : sort -key kann in XPath Ausdrücken von der sort () Funktion aufgerufen werden
- xsl : for -each -group erlaubt Gruppierung
- Zugriff auf aktuelle Gruppe mit current - group () Funktion

Beispiel Sortieren und Gruppieren

Ausgabe aller Elementnamen eines Dokuments

```
<xsl:variable name="all nodes">
  <xsl:for-each-group select="//*" group-by="name()">
    <xsl:sort select="name()"/>
    <xsl:value-of select="name(current-group())"/>
    <xsl:if test="position() != last()">, </xsl:if>
  </xsl:for-each-group>
</xsl:variable>
```

Benutzerdefinierte Funktionen

- neues Top Level Element `xsl : functi on`
- Funktionsname muss durch eigenen Namensraum abgegrenzt werden
- Aufruf innerhalb XPath-Ausdruck möglich
- deklarierte Parameter müssen übergeben werden, Vorbelegung nicht möglich
- Funktionswert muss via `xsl : resul t` zurückgegeben werden

Beispiel benutzerdefinierte Funktion

- **Namespace Declaration:**
`xml ns: cr="http://www.xml-web.de/ChangeRequests"`
- **Funktion:**

```
<xsl: function name="cr: href-fragment">
  <xsl: param name="teststring"/>
  <xsl: result>
    <xsl: value-of select="$teststring"/>
  </xsl: result>
</xsl: function>
```
- **Aufruf:**

```
<xsl: variable name="test"
  select="cr: href-fragment (@xlink: href)"/>
```


Reguläre Ausdrücke

- XPath-Funktionen `matches()`, `replace()` und `tokenize()`
- neues Element `xsl:analyze-string`
 - enthält `xsl:matching-substring`
 - oder `xsl:non-matching-substring`
- Adressierung von Teilen eines Musters mit `regex-group()` Funktion

Beispiel Nutzung reguläre Ausdrücke

Konvertierung englische in deutsche Daten:

```
<xsl:analyze-string select="current-date()"
  regex="([0-9]+)-([0-9]+)-([0-9]+)">
  <xsl:matching-substring>
    <xsl:number value="regex-group(2)" format="01"/>
    <xsl:text>.</xsl:text>
    <xsl:number value="regex-group(1)" format="01"/>
    <xsl:text>.</xsl:text>
    <xsl:number value="regex-group(3)" format="0001"/>
  </xsl:matching-substring>
</xsl:analyze-string>
```

Mehrere Ausgabedokumente/Formate

- Unterstützung mehrerer *Result Trees* neben dem *Principal Result Tree*
- Benennung von Ausgabeformaten in erweitertem `xsl : output` Element
- Referenz auf benanntes Ausgabeformat aus neuem Element `xsl : resul t-document`

Beispiel zusätzlicher *Result Tree*

- Deklaration Ausgabeformat

```
<xsl : output name="glossary-format" method="xml " />
```

- Serialisierung

```
<xsl : resul t-document href="{ $targetfn }"  
  format="glossary-format" >  
  <xsl : copy-of sel ect="document ($srcdoc) // * [ @i d=$desti d]" />  
</xsl : resul t-document >
```

Zusammenfassung

- Übung macht den Meister
 - weniger Code, aber schwieriger zu schreiben
 - Umgewöhnung auf rekursive Algorithmen
- sehr nützliches Tool für den Umgang mit XML
- XSLT 2.0 wird 2005 aktuell
 - XSLT selber wird sich weniger stark ändern
 - XPath 2.0 wird XPath 1.0 stark erweitern