

XML Vorlesung FH Aargau, SS 2006

XSL Transformation (XSLT) Teil II

Erik Wilde

19.6.2006

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

19.6.2006

XML Vorlesung FHA SS 2006

1

Übersicht

- Conflict Resolution
- Variablen und Parameter
- Sortieren
- Erzeugen des Ausgabebaumes
- XPath Erweiterungen um XSLT-Funktionen
 - Zugriff auf weitere XML-Dokumente

19.6.2006

XML Vorlesung FHA SS 2006

2

Conflict Resolution

- mehrere Templates können zutreffen
 - gleiche Match Patterns
 - Match Patterns mit Schnittmengen
 - Schnittmengen sind nicht immer leicht erkennbar
- streng genommen treten immer Konflikte auf
 - *Built-In Template Rules* matchen auf alles
 - jede eigene Template Rule erzeugt einen Konflikt
- Conflict Resolution als Entwicklungsmethode
 - beginnen mit eher allgemeinen Template Rules
 - schrittweise Verfeinerung im Laufe der Entwicklung

19.6.2006

XML Vorlesung FHA SS 2006

3

Auswahl einer Template Rule

1. alle Template Rules mit einem `match` Attribut
 - *Named Templates* werden ausgeschlossen
2. alle Template Rules mit dem gleichen `mode`
 - wird im `<xsl:apply-templates>` angegeben
3. das *Match Pattern* muss zutreffen
4. falls mehrere Match Patterns zutreffen
 - selektiere die Rule mit höchster *Import Precedence*
5. falls mehrere Import Precedences gleich sind
 - selektiere die Rule mit höchster *Priority*
6. resultierende Template Rule wird ausgeführt

19.6.2006

XML Vorlesung FHA SS 2006

4

Conflicts I: Import Precedence

- Import Precedence
 - Priorität aufgrund der Herkunft einer Template Rule
 - bestimmt durch `<xsl:import>`
- XSLT definiert einen *Import Tree*
 - alle `<xsl:include>` werden aufgelöst
 - alle `<xsl:import>` ergeben den Import Tree
- Stellung im Import Tree ist ausschlaggebend
- Built-in Template Rules
 - implizit als erstes importiert im Top-Level Stylesheet
 - auf diese Weise minimale Import Precedence

19.6.2006

XML Vorlesung FHA SS 2006

5

Conflicts II: Priority

- Priority (beliebiger numerischer Wert)
 - definiert im `priority` Attribut von `<xsl:template>`
 - höherer Wert ist eine höhere Priorität
 - falls nicht, definiert der XSLT Prozessor einen Wert
 - dieser liegt immer zwischen -0.5 und 0.5
- Priority Berechnung des XSLT Prozessors
 - Berechnung auf Grund des *Match Patterns*
 - `node()`, `text()` und `*`: -0.5
 - Namen mit Namespace Prefix: -0.25
 - Namen ohne Namespace Prefix: 0.0
 - alle anderen Match Patterns: 0.5

19.6.2006

XML Vorlesung FHA SS 2006

6

Conflicts III: kein Unterschied

- Import Precedence und Priority gleich
 - d.h. definiert im gleichen Style Sheet
 - abgesehen vom (textuell orientierten) `<xsl:include>`
 - diese Situation tritt immer im "eigenen Projekt" auf
- Standard erlaubt zwei Verhalten
 - entweder Fehlererzeugung durch den Prozessor
 - oder die letzte der Regeln wird genommen
- d.h. abhängig vom verwendeten Prozessor
 - also diesen Fall besser vermeiden
 - bei möglichen Konflikten explizite Prioritäten angeben

19.6.2006

XML Vorlesung FHA SS 2006

7

Variablen

- Zuweisung eines Wertes
 - mittels des select Attributs
 - `<xsl:variable name="name" select="'dret'"/>`
 - oder der Inhalt des `<xsl:variable>` Elements
 - komplette Template Rule Bodies sind als Inhalt erlaubt
- globale und lokale Variablen
 - in beiden Fällen ein fester Wert pro Ausführung
 - global: pro Ausführung des Stylesheets
 - lokal: pro Ausführung des Kontexts
- Variablen können nicht verändert werden
 - aber sie variieren pro Ausführung

19.6.2006

XML Vorlesung FHA SS 2006

8

Verwendung von Variablen

1. Speicherung von häufig verwendeten Werten
 - machen ein Stylesheet übersichtlicher
 - machen die Ausführung u.U. effizienter
2. Speicherung von Kontext
 - Erhalten eines Kontexts in einem neuen Kontext
 - meistens bei Iterationen mit `<xsl:for-each>`
3. Speicherung eines *Temporary Tree*
 - erlaubt komplexe Zwischenresultate
 - Wiederverwendung eines Result Tree Teiles
 - Anwendung von XPaths (nur XSLT 1.1)

19.6.2006

XML Vorlesung FHA SS 2006

9

Node Sets und Variablen

- XSLT 1.0 definiert *Result Tree Fragments*
 - eine Variable kann einen Baum enthalten
 - nur einfache Operationen sind erlaubt
 - solche, die auf einem string erlaubt sind (also nicht `$a/b`)
- XSLT 1.1 kennt keine *Result Tree Fragments*
 - eine Variable kann einen Baum enthalten
 - normale XPaths können darauf angewendet werden
 - z.B. `$variable//name[vorname='erik']`
 - sehr praktisch in vielen Situationen

19.6.2006

XML Vorlesung FHA SS 2006

10

Parameterübergabe

- Parameter übergeben Werte an Funktionen
 - Parameter des Stylesheets
 - `<xsl:param>` Element als Kind von `<xsl:stylesheet>`
 - Parameter von Templates
 - `<xsl:param>` Element als Kind von `<xsl:template>`
- Default-Wert kann angegeben werden
 - wird verwendet, falls kein Wert angegeben wird
 - falls kein Default und kein Wert: Empty String
- keine Typen (wie Variablen)
 - keine Typüberprüfung möglich
- Referenzierung wie eine Variable (`$name`)

19.6.2006

XML Vorlesung FHA SS 2006

11

Parameter für Stylesheets

- `<xsl:param>` direkt im `<xsl:stylesheet>`
- Parameter beim Stylesheet-Aufruf sind erlaubt
 - Übergabe-Art ist abhängig vom Produkt
 - oft in Form von *Command Line Arguments*
 - ebenfalls gebräuchlich sind *Environment-Variablen*
- Parameter sind Name/Value-Paare
 - entsprechender Parameter muss deklariert sein
 - falls nicht angegeben, wird Default-Wert genommen
- Parameter können beliebigen Typ haben
 - falls vom Interface unterstützt
 - kann bei einem API komplette Dokumente umfassen

19.6.2006

XML Vorlesung FHA SS 2006

12

Parameter für Templates

- `<xsl:param>` im `<xsl:template>`
- Reihenfolge Aufruf/Deklaration ist unwichtig
 - Identifikation über den Namen
- `<xsl:with-param>` setzt einen Parameter
 - wird beim Aufruf des Templates angegeben
 - `<xsl:call-template>` oder `<xsl:apply-templates>`
 - spezifiziert Name und Wert (`select` oder Inhalt)

```
<xsl:template name="loop">
  <xsl:param name="count"/>
  <xsl:call-template name="loop">
    <xsl:with-param name="count" select="$count + 1"/>
  </xsl:call-template>
</xsl:template>
```

19.6.2006

XML Vorlesung FHA SS 2006

13

Sortieren (I)

- Sortieren ist eine häufige Aufgabe
 - XSLT ist für die Aufbereitung von Dokumenten
 - Verzeichnisse werden häufig sortiert
- XSLT unterstützt Sortieren nach mehreren Keys
 - Verwendung mehrerer `<xsl:sort>` Elemente
 - z.B. Sortieren nach Vor-/Nachname

```
<xsl:for-each select="referent">
  <xsl:sort select="name"/>
  <xsl:sort select="vorname"/>
  ...
</xsl:for-each>
```

19.6.2006

XML Vorlesung FHA SS 2006

14

Sortieren (II)

- Sortieren kann gesteuert werden
 - die Sortier-Ordnung: `order`
 - `ascending` (Default) oder `descending`
 - die Case-Order: `case-order`
 - `upper-first` oder `lower-first`
 - die Sprachpräferenz: `lang`
 - Language Code (z.B. `de-CH`)
 - die Sortier-Methode: `data-type`
 - `text` (Default) oder `number` oder Selbstdefiniertes
 - einige Sortierregeln bleiben vom Standard undefiniert
 - Ergebnisse hängen u.U. vom XSLT Prozessor ab

19.6.2006

XML Vorlesung FHA SS 2006

15

Sortieren mit Templates

- `<xsl:sort>` in `<xsl:apply-templates>`
- Abarbeitung durch den XSLT Prozessor
 - Selektion eines Node Set (`select` Attribut)
 - falls nicht vorhanden, alle Kinder des *Current Node*
 - Anwendung aller `<xsl:sort>` Anweisungen
 - Aufruf des Templates (u.U. mit Parametern)
 - es können unterschiedliche Templates zum Einsatz kommen
- gemäss der Programmierphilosophie
 - eher Einsatz bei permissiven Schemas
 - verschiedene Node-Arten werden gemeinsam sortiert

19.6.2006

XML Vorlesung FHA SS 2006

16

Sortieren in Schleifen

- `<xsl:sort>` in `<xsl:for-each>`
- Abarbeitung durch den XSLT Prozessor
 - Selektion eines Node Set (`select` Attribut)
 - Anwendung aller `<xsl:sort>` Anweisungen
 - falls keine vorhanden, Abarbeitung in *Document Order*
 - Ausführung des `<xsl:for-each>` *Template Body*
 - gleicher Code für alle selektierten Nodes
- gemäss der Programmierphilosophie
 - eher Einsatz bei restriktiven Schemas
 - Sortierung einer Art von Nodes

19.6.2006

XML Vorlesung FHA SS 2006

17

Ausgaben in XSLT: Details

- Text (bereits behandelt)
- Erzeugen von Result Nodes
 - Literal Result Elements (bereits behandelt)
 - Element Nodes und Attribute Nodes
- Kopieren der Eingabe
- Generieren von Text
 - durch Erzeugen von Werten (bereits behandelt)
 - Attribute Value Templates (bereits behandelt)
- Generieren von Nummern

19.6.2006

XML Vorlesung FHA SS 2006

18

Result Nodes: Elemente

- `<xsl:element>` erzeugt ein Element
 - der Name kann berechnet werden
 - wird eher selten benutzt
 - Literal Result Elements sind besser leserlich

Result Nodes: Attribute

- `<xsl:attribute>` erzeugt ein Attribut
 - Attribut ist einem Element zugeordnet
 - Element darf noch keine Child Nodes haben
 - andernfalls wird das Attribut ignoriert
- einige Vorteile gegenüber Attributen
 - der Name kann berechnet werden
 - das Attribut wird nicht immer erzeugt
 - die Werteberechnung ist kompliziert

Ausgabe: Result Nodes

- verschiedene Arten, Nodes zu erzeugen
- XSLT erzeugt nicht Markup, sondern einen Tree
 - wichtig als Grundlage für Überlegungen

```
<xsl:template match="/">
  <xsl:element name="Element1">
    <xsl:attribute name="Attr1">42</xsl:attribute>
    <xsl:element name="Element2">
      <xsl:attribute name="Attr2">43</xsl:attribute>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

Ausgabe: Kopieren der Eingabe

- Verhältnis von Ein- und Ausgabe in XSLT
 - je nach Anwendung sehr verschieden
 - u.U. komplette Transformation
 - u.U. wenig Änderung, z.B. Extraktion eines Teils
- XSLT transformiert XML nach XML
 - Eingabe zur Ausgabe kopieren
 - Kopieren eines einzelnen Knotens
 - Kopieren eines kompletten Unterbaums

Kopieren eines Nodes

- `<xsl:copy>` kopiert den Current Node
 - keine Rekursion (kopiert alleine den Current Node)
 - rekursives Kopieren mit `<xsl:copy-of>`
- Verhalten richtet sich nach dem Node Type
 - Root: wird ignoriert
 - Element: wie eine `<xsl:element>` Anweisung
 - Attribut: wie eine `<xsl:attribute>` Anweisung
 - Text: erzeugt einen Text Node
 - PI: erzeugt einen PI Node
 - Comment: erzeugt einen Comment Node
- Inhalt wird ausgeführt bei Root und Element

19.6.2006

XML Vorlesung FHA SS 2006

23

Kopieren eines Node Trees

- `<xsl:copy-of>` kopiert Node Tree
 - Rekursion ausgehend vom `select` Attribut
 - es können auch Variablen angegeben werden
 - nicht-rekursives Kopieren mit `<xsl:copy>`
- Verhalten je nach Typ des `select`
 - Node Set wird in Document Order abgearbeitet
 - alle Nodes werden einzeln rekursiv abgearbeitet
 - andere Typen (String, Number oder Boolean)
 - Effekt identisch zu `<xsl:value-of>`
 - Konvertierung in String und Erzeugung eines Text Node

19.6.2006

XML Vorlesung FHA SS 2006

24

XSLT Funktionen in XPath (I)

- `document()` Funktion liest Dokument
 - kann als *Node Set* behandelt werden (Location Path)
 - einzige Möglichkeit für weitere Eingaben in XSLT
- `key()` als Generalisierung von `ID/IDREF(S)`
 - ein Key besteht aus drei Komponenten
 - ein Node, dem der Key zugeordnet ist
 - der Name des Keys
 - der Wert des Keys (ein String)
 - `<xsl:key>` deklariert einen Key
 - Zuweisung einer Menge von Node/Name/Wert Triples
 - `key()` ermöglicht den Zugriff auf Triples
 - Eingabe sind Name und Wert, Resultat ist der Node

XSLT Funktionen in XPath (II)

- `format-number()` für formatierte Zahlen
 - Verwendung eines Patterns für die Formatierung
 - Referenz auf JDK 1.1 DecimalFormat Class
 - Verwendung mit `<xsl:decimal-format>`
- `current()` liefert den Context Node
 - notwendig in XPath Location Paths
 - Verwendung in Prädikaten von Location Steps
 - `<xsl:if test="//topic[@TID=current()/@TID]">`
- `generate-id()` generiert eine eindeutige ID
 - ergibt einen eindeutigen XML Name (ID Typ)
 - nur konsistent für eine Ausführung eines Style Sheets

XSLT Funktionen in XPath (III)

- `unparsed-entity-uri()`
 - liefert die URI eines unparsed Entity
 - Referenzierung über den Namen
 - nicht über den XPath Node Tree erreichbar
 - aber Teil des Information Set (eigenes Information Item)
- `system-property()`
 - liefert Eigenschaften des XSLT Prozessors
 - drei notwendige vordefinierte Eigenschaften
 - `xsl:version`, `xsl:vendor`, `xsl:vendor-url`
 - beliebige weitere Eigenschaften (prozessorspezifisch)

Zusammenfassung

- XSLT bietet einfache Grundlagen
 - XPath mit wenigen Erweiterungen
 - einfache Kontrollkonstrukte
- Anwendung ist prinzipiell einfach
 - aber erfordert etwas Gewöhnung
 - kann u.U. etwas Trickserei erfordern