

XSL Transformations (XSLT) Teil I

Erik Wilde

12.6.2006

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

12.6.2006

XML Vorlesung FHA SS 2006

1

Übersicht

- 8³⁰ – 9³⁰: Test (XML Schema)
- Herkunft von XSLT
- XSLT aus der Ferne betrachtet
- Template Rules
 - Built-in Template Rules
 - XSLT Processing Model und Patterns
- Ein- und Ausgaben mit XSLT
- Anweisungen in XSLT
 - einige ausgewählte Anweisungen

12.6.2006

XML Vorlesung FHA SS 2006

2

XSL Transformations (XSLT)

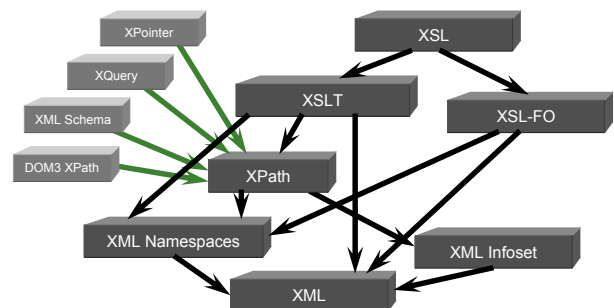
- ursprünglich Teil des XSL Standards
 - jetzt per Referenz in den Standard integriert
 - orientiert an Scheme, einem Lisp-Dialekt
- Potential wurde erkannt
 - Trennung von XSL in XSLT und XSL-FO
 - XSLT transformiert in *XSL Formatting Objects*
 - XSL-FO werden zur Darstellung benutzt
- XSLT wurde weiter geteilt
 - XPath für die Selektion von Teilen eines Dokuments
 - "der Rest", die Kontrollstrukturen
- besserer Name: *XML Transformation Language*

12.6.2006

XML Vorlesung FHA SS 2006

3

"Übersicht" XSL-Standards



12.6.2006

XML Vorlesung FHA SS 2006

4

Style Sheets für XML Dokumente

- separater W3C Standard (06/99)
 - Verbindung eines XML Dokuments mit Style Sheet
 - entspricht dem <LINK> Element in HTML
- definiert spezielle Processing Instruction
 - `<?xml-stylesheet ... ?>`
 - ignoriert falls nicht unterstützt
- definiert Attribute für
 - Typ (MIME) und Referenz (URI)
 - Titel, Medientyp, Character Set und Alternate
 - Semantik genau gleich wie für HTML Style Sheets

12.6.2006

XML Vorlesung FHA SS 2006

5

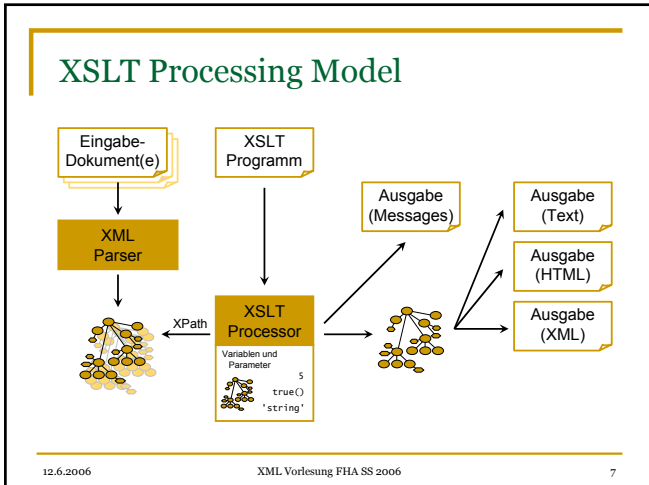
XSLT aus der Ferne betrachtet

- Eingabe ist ein XML-Dokument
 - etwas genauer betrachtet ein XPath Node Tree
 - *Whitespace Stripping* als erster Schritt
 - je nachdem wie verlangt (Default: kein Stripping)
- Transformation als Ausführung des XSLT
 - beliebige Komplexität der Abarbeitung
- Ausgabe ist XML, HTML oder Text
 - XML ist der Normalfall (erlaubt Konkatenation)
 - XML Dokumente und *External General Parsed Entities*
 - HTML als populäres Präsentationsformat
 - Text ohne Markup-Struktur

12.6.2006

XML Vorlesung FHA SS 2006

6



- ### Funktionales Programmieren
- XSLT ist eine funktionale Programmiersprache
 - für viele ein neues Konzept
 - bekannt aus Sprachen wie Lisp und Scheme
 - alles ist eine Funktion
 - Funktionen sind *side effect free*
 - hängen einzig von den Eingabeparametern ab
 - können in beliebiger Reihenfolge aufgerufen werden
 - gute Unterstützung für inkrementelles Rendering
 - keine (echten) Variablen in der Sprache
 - `<xsl:variable>` kann nicht geändert werden
- 12.6.2006 XML Vorlesung FHA SS 2006 8

first.xsl

XSLT: My First Stylesheet!

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

- was an diesem Programm auffällt
 - es ist ein XML-Dokument
 - es benutzt XML Namespaces
 - es ist eine leere Hülle (nur ein Container-Element)
 - und es funktioniert trotzdem!

12.6.2006 XML Vorlesung FHA SS 2006 9

- ### "First Stylesheet" unter der Lupe
- Ergebnis der Anwendung auf ein Dokument
 - der "Text" des Dokuments erscheint
 - Text-Inhalt der Elemente, aber nicht der Attribute
 - funktioniert bei beliebigen Dokumenten
 - Default-Verhalten
 - ungewöhnlich für eine Programmiersprache
 - praktisch für simple "Default-Formatierung"
 - inkrementelle Entwicklung einfach möglich
 - beginnen mit einem leeren Stylesheet
 - stufenweise Verfeinerung in ausführbaren Schritten
- 12.6.2006 XML Vorlesung FHA SS 2006 10

second.xsl

XSLT: My Second Stylesheet!

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:template match="*">
    ( Element: <xsl:value-of select="local-name()"/>
      <xsl:apply-templates select="* | @*" /> )
  </xsl:template>
  <xsl:template match="@*">
    Attribute: <xsl:value-of select="local-name()"/>
  </xsl:template>
</xsl:stylesheet>
```

12.6.2006 XML Vorlesung FHA SS 2006 11

- ### Resultat des "Second Stylesheet"
- ```
(Element: topicmap
 (Element: head
 (Element: title)
 (Element: author))
 (Element: body
 (Element: topics
 (Element: topic Attribute: TID
 (Element: derived-from Attribute: template)
 (Element: name)
 (Element: text))
 (Element: topic Attribute: TID ...
```
- Whitespace nachträglich von Hand verändert!
- 12.6.2006 XML Vorlesung FHA SS 2006 12

## "Second Stylesheet" unter der Lupe

- Steuerung der Ausgabe möglich
  - normalerweise XML als Ausgabe
  - Text-orientierte Ausgabe ebenfalls erlaubt
- Programmsteuerung durch *Template Rules*
  - Rekursion als Normalfall
  - Selektion der Template Rules durch XSLT Prozessor
  - Ausführung durch das Dokument gesteuert
- inkrementelle Entwicklung
  - ohne die zweite Template Rule ebenfalls lauffähig
  - aber: Default-Verhalten in diesem Fall ungünstig

12.6.2006

XML Vorlesung FHA SS 2006

13

helloworld.xml

## XSLT: Hello World!

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

 <xsl:template match="/">
 <xsl:text>Hello world!</xsl:text>
 </xsl:template>

</xsl:stylesheet>
```

- Abarbeitung beginnt beim Root Node
- Ausgabe eines Text Nodes
- Ende der Abarbeitung

12.6.2006

XML Vorlesung FHA SS 2006

14

## 1. Übung

- erzeugen Sie ein Stylesheet
  - bei "Null" beginnen
  - oder die Vorlagen aus den Unterlagen nehmen:
    - <http://dret.net/lectures/xml-fha-ss04/xslt-files/>
- verbinden Sie Ihr XML mit dem Stylesheet
  - per `xml-stylesheet` PI im XML
  - oder über Kommandozeile oder Tools Support
- führen Sie das Stylesheet aus
  - mit eingebautem XSLT Prozessor
  - mit dem Saxon XSLT Prozessor (Java Programm)
    - <http://saxon.sourceforge.net>

12.6.2006

XML Vorlesung FHA SS 2006

15

## XSLT als Programmiersprache

- Transformation von XML für die Darstellung
  - gedacht für den Einsatz im Browser
    - limitierte Ausführungsumgebung
    - keine Kontrolle der Ausführungsumgebung
  - spezialisiert auf ein Anwendungsgebiet
- XSLT wird vielfach falsch eingesetzt
  - missverstanden als general purpose Sprache
- XSLT und Software Engineering
  - viele implizite Typkonvertierungen
  - Implementierungsfreiheiten für XSLT Prozessoren
  - erfordert viel Aufwand und Eigendisziplin

12.6.2006

XML Vorlesung FHA SS 2006

16

## Built-in Template Rules

- fest eingebaut in jeden XSLT Prozessor
  - definiert im XSLT Standard
  - definieren das Default-Verhalten
    - deshalb erzeugt ein "leeres" XSLT eine Ausgabe
- streng genommen gibt es also immer Konflikte
  - ausser für Namespace Nodes
    - können nicht durch ein Pattern selektiert werden
  - Conflict Resolution ist essentiell in XSLT!
- Built-in Rules werden als importiert behandelt
  - vor allen anderen `<xsl:import>` Anweisungen
  - deshalb eine geringere *Import Precedence*

12.6.2006

XML Vorlesung FHA SS 2006

17

## Built-in Template Rules (I)

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

```
<xsl:apply-templates/>
</xsl:template>
```
- definiert Regel für Root und Element Nodes
- weitere Suche nach Templates
- rekursives Abarbeiten aller Nodes
  - allerdings nur die Children der Nodes
    - Verhalten bei `<xsl:apply-templates>` ohne `select`
  - d.h. Attribute und Namespaces nicht

12.6.2006

XML Vorlesung FHA SS 2006

18

### Built-in Template Rules (II)

- `<xsl:template match="*/|" mode="m">  
 <xsl:apply-templates mode="m"/>  
</xsl:template>`
- Äquivalent für die erste Regel, aber mit Modes
  - definiert für alle Modes im Stylesheet
- erlaubt rekursives Abarbeiten mit Modes
  - der Mode bleibt dabei erhalten

12.6.2006

XML Vorlesung FHA SS 2006

19

### Built-in Template Rules (III)

- `<xsl:template match="text()|@">  
 <xsl:value-of select="."/>  
</xsl:template>`
- definiert Regel für Text und Attribute Nodes
  - erzeugt den Text des jeweiligen Nodes
- zu beachten bei dieser Regel
  - Text Nodes sind Kinder von Elementen
    - werden durch die Built-in Template Rules selektiert
  - Attribute Nodes sind nicht Kinder von Elementen
    - werden durch die Built-in Template Rules nicht selektiert

12.6.2006

XML Vorlesung FHA SS 2006

20

### Built-in Template Rules (IV)

- `<xsl:template match="processing-instruction()|comment()"/>`
- Regel für PI und Comment Nodes
  - leeres Template → Ignorieren der Nodes

12.6.2006

XML Vorlesung FHA SS 2006

21

### Built-in Template Rules (V)

- es gibt kein Pattern für Namespaces Nodes
  - Konsequenz der Definition von Patterns
    - nur Child und Attribute Axis sind erlaubt
  - deshalb kann es auch keine Template Rule geben
- eine "eingebaute" Rule ist dennoch definiert
  - aber nicht als XSLT darstellbar
  - sie definiert, nichts zu tun (wie bei PI und Comment)
- kann nicht überschrieben werden
  - Namespace-Behandlung durch Templates unmöglich
  - falls notwendig, `<xsl:for-each>` nehmen

12.6.2006

XML Vorlesung FHA SS 2006

22

### Templates Rules

- Templates werden auf zwei Arten aktiviert
  - über das *Match Pattern* (match Attribut)
  - als *Named Template* (name Attribut)
  - Templates können auch beide Attribute tragen
- Templates können Parameter haben
  - diese werden beim Aufruf gesetzt
  - sind (neben dem Context Node) die Eingabe
  - werden als `<xsl:param>` deklariert (kein Typ!)
- enthalten einen *Template Rule Body*
  - ähnlich dem Block-Konzept in anderen Sprachen
  - wird an weiteren Stellen verwendet

12.6.2006

XML Vorlesung FHA SS 2006

23

### Patterns

- sind eine Untermenge von XPath
  - müssen als Resultat ein Node Set liefern
  - werden von XSLT definiert (nicht von XPath)
- werden an vier Stellen verwendet
  - `<xsl:template match="...">`
  - `<xsl:key match="...">`
  - `<xsl:count from="..." to="...">`
- Menge von Selektionskriterien für Nodes
  - werden die Bedingungen erfüllt, matcht der Node
  - andernfalls nicht (das Pattern trifft nicht zu)

12.6.2006

XML Vorlesung FHA SS 2006

24

## XSLT Processing Model

- eine Liste von *Source Nodes* ergibt das Resultat
- Abarbeitung beginnt mit *Root Node* als *Source Node*
- die *Source Node* Liste wird abgearbeitet, und die *Result Tree Fragments* werden an den *Result Tree* angehängt
- die folgenden Schritte werden wiederholt
  1. für jeden *Source Node* werden alle *Template Rules* mit matching *Patterns* gesucht, das beste wird ausgeführt mit dem *Source Node* als *Current Node* (dies bestimmt den Kontext)
  2. Templates selektieren oft weitere *Source Nodes*
    - diese werden in die *Source Node* Liste eingefügt
  3. der Ablauf von Matching, Ausführung und Selektion wird rekursiv fortgesetzt, bis die *Source Node* Liste leer ist

12.6.2006

XML Vorlesung FHA SS 2006

25

## Template Rule Anwendung

- Templates werden auf zwei Arten aufgerufen
  - `<xsl:apply-templates>` selektiert Nodes
    - arbeitet die Nodes mit neuem Kontext ab
  - `<xsl:call-template>` für ein *Named Template*
    - keine Änderung des Kontext
- `<xsl:apply-templates>`
  - optional können die Nodes sortiert werden
  - Ausführung ist durch das Dokument bestimmt (push)
- `<xsl:call-template>`
  - sehr ähnlich einem Prozeduraufruf
  - Ausführung durch das Stylesheet bestimmt (pull)

12.6.2006

XML Vorlesung FHA SS 2006

26

## Eingaben in XSLT

- Eingabe ist immer ein XPath Node Tree
  - wird aus Infoset erzeugt (d.h. Namespace-compliant)
- XSLT Prozessor arbeitet auf dem Node Tree
  - XSLT sagt nichts über die Herkunft des Trees
  - oftmals ein vorgeschalteter XML Parser
  - kann auch ein synthetisiertes DOM sein
- u.U. kann der Parser wichtig sein
  - z.B. Validierung (ID/IDREF, Defaults)
  - z.B. Whitespace Handling (Stripping von Text Nodes)
  - Steuerung des Parsers manchmal notwendig

12.6.2006

XML Vorlesung FHA SS 2006

27

## Ausgaben in XSLT

- Ziel abhängig von der Anwendung
  - oftmals ein XML-Dokument (Result Tree)
  - definiert durch `<xsl:output>`
- Varianten für die Ausgabe
  - Text-orientierte Ausgabe möglich
    - falls nicht nur XML verwendet wird (z.B. Comma separated)
  - HTML-Ausgabe möglich
    - Ausgabe ist kein XML-Dokument
    - XHTML als Methode erst in XSLT 2.0 definiert
  - binäre Daten können nicht erzeugt werden
    - nicht das Anwendungsgebiet von XSLT

12.6.2006

XML Vorlesung FHA SS 2006

28

## Ausgaben in XSLT: Result Tree

- Text
- Erzeugen von Result Nodes
  - Literal Result Elements
  - Element Nodes und Attribute Nodes (später mehr)
  - Named Attribute Groups (später mehr)
- Kopieren der Eingabe (später mehr)
- Generieren von Text
  - durch Erzeugen von Werten
  - Attribute Value Templates (AVTs)
- Generieren von Nummern (später mehr)

12.6.2006

XML Vorlesung FHA SS 2006

29

## Ausgabe: Text

- Text ist in Template Rule Bodies erlaubt
  - Text Nodes oder `<xsl:text>` im Stylesheet
  - Inhalt wird als Text Node in den Result Tree kopiert
- *Whitespace Stripping* vor XSLT Abarbeitung
  - XSLT Prozessor liest XSLT und XML Dokument
    - Whitespace Text Nodes im XSLT werden gelöscht
    - Whitespace Text Nodes im XML bleiben erhalten
      - Steuerung im XSLT möglich (auf Basis der XML Elemente)
  - der gestrippte Node Tree wird abgearbeitet
- Entities bleiben nicht erhalten
  - verschwinden schon beim Parsen zum Node Tree

12.6.2006

XML Vorlesung FHA SS 2006

30

## Result Nodes: Literal Result Elements

- Elemente, die nicht im XSLT Namespace sind
  - XSLT ist immer ein XML Dokument
  - aber fast immer nicht gemäss dem XSLT Schema
    - es gibt kein normatives Schema im XSLT Standard
- erzeugen Result Nodes
  - einfache Einbettung von XSLT und anderem Schema
  - erzeugen einfach lesbares XSLT
- können XSLT-Attribute tragen
  - Steuerung der Ausgabe, Erzeugung von Attributen
- Einschränkungen bei der Benutzung
  - Element-Name ist hard coded im XSLT

12.6.2006

XML Vorlesung FHA SS 2006

31

xpath.xsl

## Ausgabe: Erzeugen von Werten

- `<xsl:value-of>` evaluiert einen XPath
  - das Resultat kommt als String in den Result Tree
  - es gibt deshalb u.U. implizite Konvertierungen
- häufige Verwendung mit XPath Location Paths

```
Topic: <xsl:value-of select="child::name"/>
 (@TID=" <xsl:value-of select="attribute::TID"/>")
<xsl:if test="child::alias or child::text">

 <xsl:if test="child::alias">
 Alias: <xsl:value-of select="child::alias"/>
 </xsl:if>
 <xsl:if test="child::text">
 Text: <xsl:apply-templates select="child::text"/>
 </xsl:if>

</xsl:if>

```

12.6.2006

XML Vorlesung FHA SS 2006

32

## Ausgabe: Attribute Value Templates

- Erzeugung von Attributen aus Expressions
  - erlaubt bei Attributen von Literal Result Elements
  - erlaubt bei Attributen von einigen XSLT Elementen
    - sehr eingeschränkte Liste, Implementierungsüberlegungen
- XPath Expression eingeschlossen in { und }
  - Auswertung gemäss normalen Regeln (mit gegebenem Kontext)
  - Konvertierung in einen String
  - das Resultat ist der Wert des Attribute Value Templates

```

 <xsl:value-of select="@email"/>

```

12.6.2006

XML Vorlesung FHA SS 2006

33

## Iterationen

- `<xsl:for-each>` erlaubt Schleifen
  - Selektion eines Node Sets
    - absolut oder relativ zum aktuellen Kontext
  - Ausführung des Template Body für alle Nodes
  - die Nodes können sortiert werden
    - `<xsl:sort>` Element(e) innerhalb des `<xsl:for-each>`
  - der Kontext wird für jeden Node neu gesetzt
    - falls alter Kontext gebraucht wird: Variable setzen
  - ähnlicher Effekt zu `<xsl:apply-templates>`
    - Pull vs. Push Processing

12.6.2006

XML Vorlesung FHA SS 2006

34

## Conditional Code

- `<xsl:if>` für einfache Bedingung (kein "else")

```
<xsl:if test="$somecondition">
 <xsl:text>$somecondition is true()</xsl:text>
</xsl:if>
```

- `<xsl:choose>` für Auswahl aus Alternativen
  - `<xsl:when>` darf wiederholt werden ("else if")
  - `<xsl:otherwise>` als letzter "else" Teil

```
<xsl:choose>
 <xsl:when test="$count > 2"><xsl:text>, and
 </xsl:text></xsl:when>
 <xsl:when test="$count > 1"><xsl:text> and
 </xsl:text></xsl:when>
 <xsl:otherwise><xsl:text> </xsl:text></xsl:otherwise>
</xsl:choose>
```

12.6.2006

XML Vorlesung FHA SS 2006

35

## Programmierphilosophie

- `<xsl:apply-templates>` für Match Patterns
- `<xsl:for-each>` für Iterationen
- Templates vs. explizite Schleifen
  - eine Frage der Methodik
  - eine Frage der Wartbarkeit
    - kann mit `<xsl:call-template>` verbessert werden
  - eine Frage der Schemas und Instanzen
    - "relational" vs. semi-structured XML
  - eine Frage der Gewöhnung
- *push vs. pull* Methodik
  - <http://www.ibm.com/developerworks/xml/library/x-xdpshpu1.html>

12.6.2006

XML Vorlesung FHA SS 2006

36

## <xsl:apply-templates>

- zweistufiger Prozess
  - Selektion von Nodes (`select` Attribut)
  - Suchen von passenden Templates (*Match Pattern*)
- vom Dokument gesteuert
  - *Push-Processing*
  - Ausführung u.U. abhängig von Imports
- besser geeignet für permissive Schemas
  - Inhalt eines Elementes stark variabel
  - Behandlung durch das Dokument gesteuert
    - dynamische Auswahl der passenden Templates

12.6.2006

XML Vorlesung FHA SS 2006

37

## <xsl:for-each>

- einstufiger Prozess
  - Selektion von Nodes (`select` Attribut)
- vom Stylesheet gesteuert
  - *Pull-Processing*
  - vorhersagbare Ausführung
- besser geeignet für restriktive Schemas
  - Inhalt eines Elementes recht klar eingegrenzt
  - Behandlung genau dieses Inhaltes
    - weniger flexibel hinsichtlich variabler Dokumente
    - mehr Kontrolle über den ausgeführten Code

12.6.2006

XML Vorlesung FHA SS 2006

38

## Zusammenfassung

- XSLT als Programmiersprache
- viele Gemeinsamkeiten mit anderen Sprachen
  - einfache Kontrollkonstrukte
  - Ein- und Ausgaben
  - sequentielle Abarbeitung der Templates
- einige gewöhnungsbedürftige Besonderheiten
  - zugrundeliegendes XML-Modell
  - Abarbeitung des Source Tree
    - Auswahl der passenden Templates
  - funktionale Sprache (Rekursion statt Iteration)

12.6.2006

XML Vorlesung FHA SS 2006

39