universität wien

Faculty of Computer Science
Workflow Systems and Technology Group

XSL Stylesheets und Transformationen

Juergen Mangler
University of Vienna

XSL

XSL = e**X**tensible **S**tylesheet **L**anguage
http://www.w3.org/Style/XSL/

Idee: Ein Stylesheet Instrument für XML (analog z.B. CSS für HTML) zwecks Präsentation und Transformation

XSL = XSLT + XPath + XSL-FO

- XSLT = XSL Transformations (Transformation von XML Dokumenten
- XPath = XML Path Language (Navigation in XML Dokumenten)
- XSL-FO = XSL Formatting Objects (geräteunabhängige Formattierung von Dokumenten)

Instrument für XML Präsentation und Transformation

XSL-FO

Definition einer Markup-Sprache für geräteunabhängige **Präsentation** von XML Dokumenten (Drucken, Bildschirm, PDF, etc.)

Mächtiger als CSS

XSLT

Instrument für die **Transformation** von XML Markup in anderes XML Markup oder andere Formate (z.B. XHTML, HTML, PDF, Text, etc.)

Wird generell für Transformationen von XML-Quelldokumenten verwendent

Verwendet XPath für Navigation und Adressierung in XML Dokumenten

Ist der wichtigste Teil von XSL

XSLT Anwendungsfall

Wir haben ein XML Dokument als Input

Wir wollen es in ein gewünschtes (anderes) Outputformat transformieren (= umordnen, auswählen, filtern, hinzufügen, ...)

Das Outputformat ist beliebig:

- XML (gleiches oder anderes Markup)
- Nicht-XML Markup, z.B. HTML
- Text (kein Markup), z.B. PDF, Plaintext

Vor allem in Webanwendungen häufiger Fall:

- Datendefinition in XML (Input)
- Datenpräsentation in HTML (Output)
- (s. Beispiel auf Folgefolien)

XSLT Beispiel: Input

Input: Beispiel Bundesregierung

```
<?xml version='1.0' encoding='UTF-8'?>
<staff organization="Bundesregierung">
    <person id="agu">
        <name>Alfred Gusenbauer</name>
        <party url="http://www.spoe.at">SPÖ</party>
    </person>
    <person id="wmo">
        <name>Wilhelm Molterer</name>
        <party url="http://www.oevp.at">ÖVP</party>
    </person>
</staff>
```

XSLT Beispiel: Output

Output soll in HTML sein und wie folgt aussehen:

```
<html>
  <head>
      <title>Bundesregierung</title>
  </head>
  <body>
      <p>Alfred Gusenbauer</p>
      <p>Wilhelm Molterer</p>
  </body>
</html>
```

XSLT Beispiel: Verarbeitung allgemein

Zwei **Schritte**:

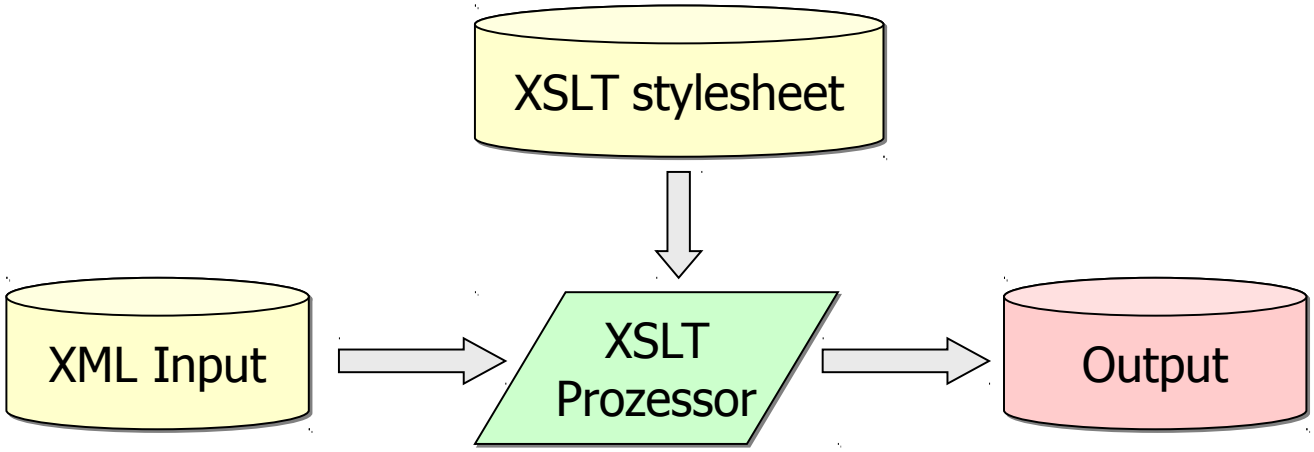
1. Definiere die Transformation mit einem XSLT Stylesheet
2. Führe die Transformation auf das Inputdokument aus mit einem XSLT Prozessor

XSLT **Prozessor**

Lädt das XSLT Stylsheet

Lädt das Inputdokument (DOM Baum oder File)

Transformiert das Inputdokument mit dem Stylesheet und liefert das Outputdokument



XSLT Beispiel: Verarbeitung in Java

```
DocumentBuilderFactory docBuildFactory =
      DocumentBuilderFactory.newInstance();
DocumentBuilder parser =
      docBuildFactory.newDocumentBuilder();
Document xml = parser.parse(new File(argv[1]));

StreamSource xslsource =
      new StreamSource(new File(argv[0]));
TransformerFactory xformFactory =
      TransformerFactory.newInstance();
Transformer transformer =
      xformFactory.newTransformer(xslsource);

DOMSource xmlsource = new DOMSource(xml);
StreamResult scrResult = new StreamResult(System.out);
transformer.transform(xmlsource, scrResult);
```

Laden des Inputdokuments

Laden des Stylesheet

Initialisieren des XSLT Prozessors mit dem Stylsheet

Transformation des Inputdokuments und Ausgabe

XSLT ist eine **deklarative** Sprache

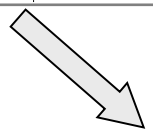
Es wird eine Menge von **Regeln** definiert, die auf das Inputdokument gematcht werden

Stylesheet besteht aus **Templates** (Schablonen) die den Regeln entsprechen und vom XSLT Prozessor verarbeitet werden

Der XSLT Prozessor ruft beim Durchgehen der Knoten des Baumes des Inputdokuments das im Stylesheet für den jeweiligen Knoten passende Template auf

Regel 1: Erzeuge aus dem Wurzelelement die HTML Grundstruktur

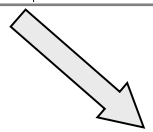
```
<?xml version='1.0' encoding='UTF-8'?>
<staff organization="Bundesregierung">
   <person id="agu">
      <name>Alfred Gusenbauer</name>
      <party url="http://www.spoe.at">SPÖ</party>
   </person>
   <person id="wmo">
      <name>Wilhelm Molterer</name>
      <party url="http://www.oevp.at">ÖVP</party>
   </person>
</staff>
```



```
<html>
   <head>
      <title>Bundesregierung</title>
   </head>
   <body>
      <p>Alfred Gusenbauer</p>
      <p>Wilhelm Molterer</p>
   </body>
</html>
```

Regel 2: Erzeuge für jedes Person-Element einen Absatz mit dem Namen der Person

```
<?xml version='1.0' encoding='UTF-8'?>
<staff organization="Bundesregierung">
  <person id="agu">
    <name>Alfred Gusenbauer</name>
    <party url="http://www.spoe.at">SPÖ</party>
  </person>
  <person id="wmo">
    <name>Wilhelm Molterer</name>
    <party url="http://www.oevp.at">ÖVP</party>
  </person>
</staff>
```

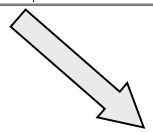


```
<html>
  <head>

<title>Bundesregierung</title>
  </head>
  <body>
    <p>Alfred Gusenbauer</p>
    <p>Wilhelm Molterer</p>
  </body>
</html>
```

Regel 2: Erzeuge für jedes Person-Element einen Absatz mit dem Namen der Person

```
<?xml version='1.0' encoding='UTF-8'?>
<staff organization="Bundesregierung">
  <person id="agu">
    <name>Alfred Gusenbauer</name>
    <party url="http://www.spoe.at">SPÖ</party>
  </person>
  <person id="wmo">
    <name>Wilhelm Molterer</name>
    <party url="http://www.oevp.at">ÖVP</party>
  </person>
</staff>
```

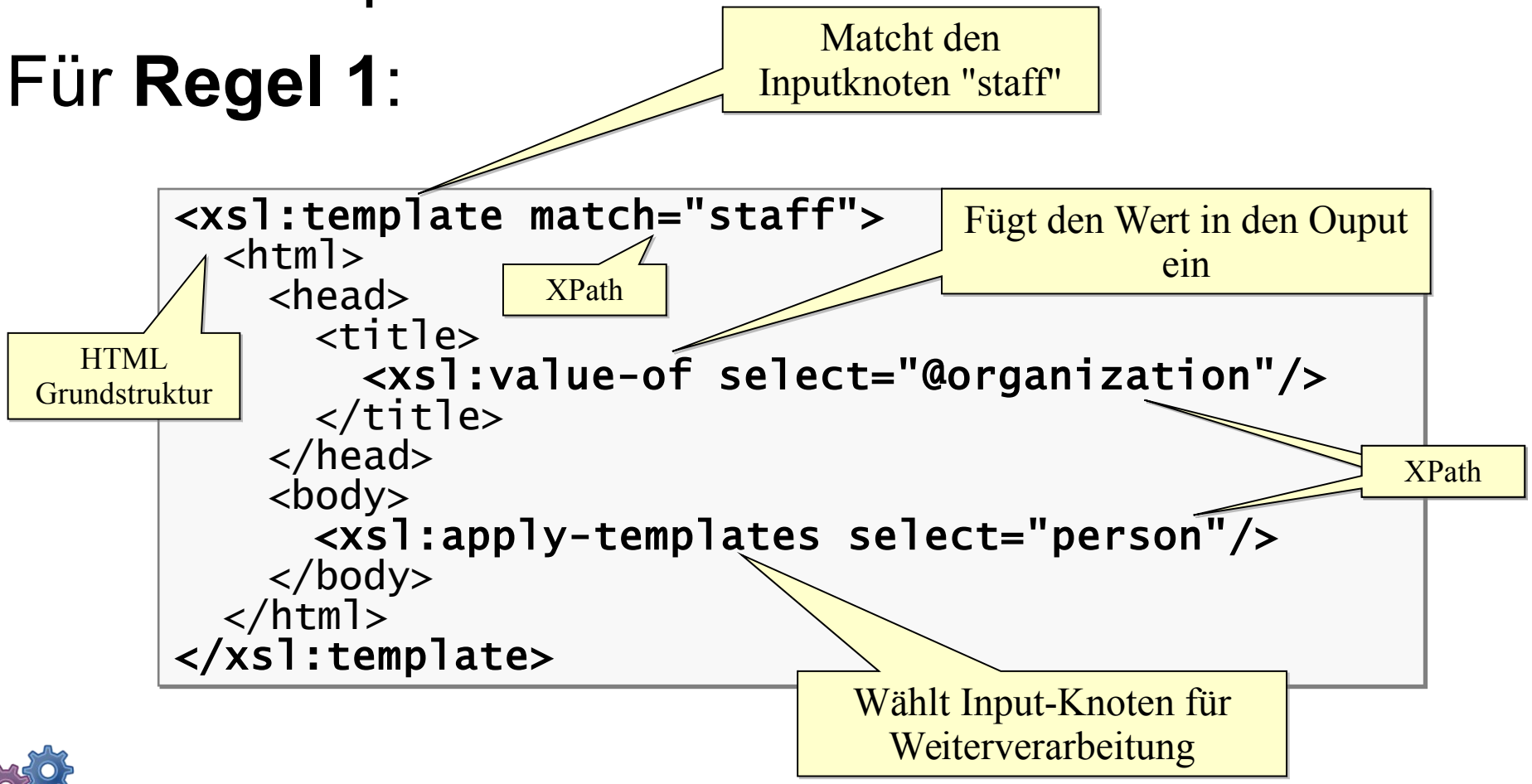


```
<html>
  <head>

<title>Bundesregierung</title>
  </head>
  <body>
    <p>Alfred Gusenbauer</p>
    <p>Wilhelm Molterer</p>
  </body>
</html>
```

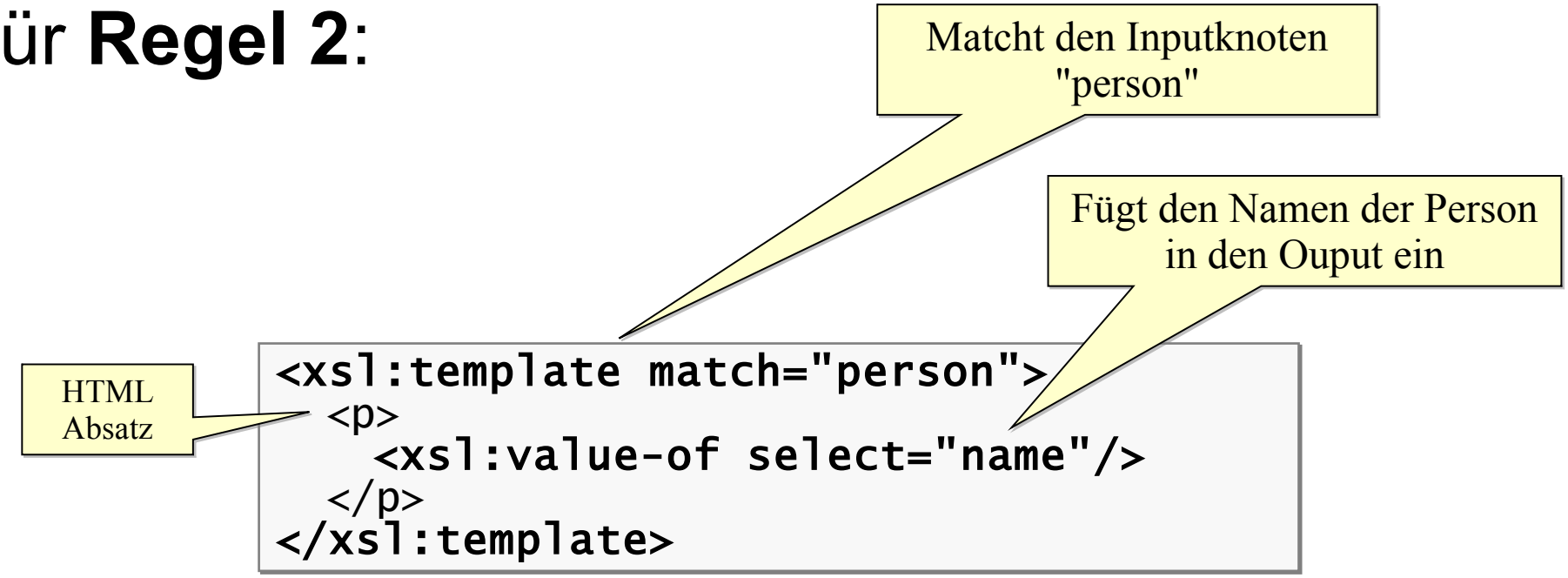
Definiere nun aus den identifizierten Regeln XSLT Templates

Für **Regel 1**:

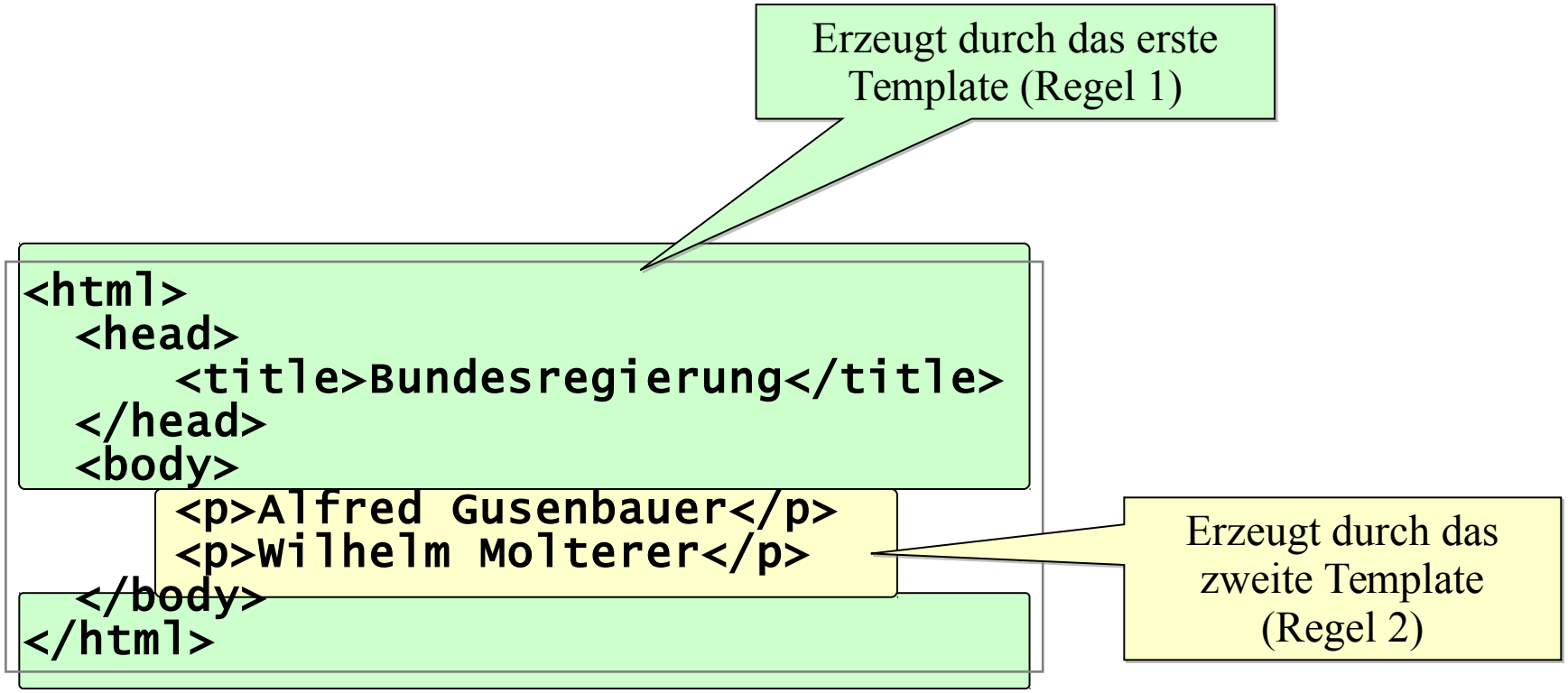


Erzeuge nun aus den identifizierten Regeln XSLT Templates

Für **Regel 2**:



XSLT Beispiel: Ergebnis



XSLT Stylsheet Deklaration

XSLT Stylesheet ist selber ein XML Dokument und verwendet den Namespace mit URI *http://www.w3.org/1999/XSL/Transform* (für diesen Namespace wird meistens das Präfix **`xsl`** verwendet)

Daher sieht das XSLT Skelett wie folgt aus (in den folgenden Folien werden wir diesen Teil weglassen und uns nur auf den Inhalt konzentrieren):

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

   ...Stylesheet Inhalt...

</xsl:stylesheet>
```

Templates

Der XSLT Prozessor **parst das XML Inputfile** und sucht für die Knoten geeignete Templates

Wird ein Template gefunden, dass für einen Knoten passt, so wird das **Template angewendet**

Die Regeln für **Template-Abarbeitung** sind recht kompliziert, daher empfiehlt es sich anfangs folgendes zu tun:

- Definiere ein Template für den Wurzelknoten
- Steuere aus diesem Template die weitere Abarbeitung

Beispiel: das Template matcht den Wurzelknoten und gibt den Wert des Attributs aus.

```xml
<A x="25">
  <B>Hallo!</B>
</A>
```
XML

```xsl
<xsl:template match="/A">
   <xsl:value-of select="@x"/>
   In Element B steht: </xsl:value-of select="B"/>
</xsl:template>
```
XSL

Ausgabe:

```
25
In Element B steht: Hallo!
```

Default Templates

Es gibt Templates, die implizit immer vorhanden sind. Deren Verhalten ist für die folgenden Knoten wie folgt:

Wurzel und **Elemente**: Wende Templates für die Kinder an (rekursiv!)

Text: Kopiere Inhalt in den Output

Attribut: Kopiere Wert in den Output

Will man Verhalten von Default Templates für bestimmte Knoten ändern (überschreiben), so kann man einfach das Template explizit im Stylesheet definieren

```xml
<A>
  <B>Hallo</B>
  <C>Welt!</C>
</A>
```
XML

```xsl
<xsl:template match="B">
   <xsl:value-of select="."/>
</xsl:template>
```
XSL

Ausgabe:

```
Hallo
Welt!
```

Warum wird auch der Inhalt von C ausgegeben?

Default Templates

Die vordefinierten Templates kann man sich wie folgt vorstellen:

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

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

Betrifft Wurzel und Kinder

Sagt, dass für die Kindelemente weitere Templates angewendet werden sollen

Betrifft Attribute und Textknoten

Das Stylesheet von der vorigen Seite sieht also implizit wie folgt aus:

```
<A>
  <B>Hallo</B>
  <C>Welt!</C>
</A>
```

XML

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

Matcht den Text von C

XSL

Default Templates

Um die Abarbeitung des Inputdokumentes zu "visualisieren", können wir z.B. folgendes "Debugging Template" verwenden:

```xml
<A>
  <B x="25">Hallo</B>
  <C y="10">Welt!</C>
</A>
```
XML

```
<xsl:template match="/">
   Wurzel,
   <xsl:apply-templates select="*|text()|@*"/>
</xsl:template>
<xsl:template match="*">
   Element <xsl:value-of select="name()"/>,
   <xsl:apply-templates select="*|text()|@*"/>
</xsl:template>
<xsl:template match="@*">
   Attribut @<xsl:value-of select="name()"/>=
   <xsl:value-of select="."/>,
</xsl:template>
<xsl:template match="text()">
   Text(&quot;<xsl:value-of select="."/>&quot;),
</xsl:template>
```

Ausgabe

```
Wurzel, Element A, Element B, Attribut @x= 25, Text("Hallo"),
Element C, Attribut @y= 10, Text("Welt!"),
```

Templates

Der XSLT Prozessor **parst das XML Inputfile** und sucht für die Knoten geeignete Templates

Wird ein Template gefunden, dass für einen Knoten passt, so wird das **Template angewendet**

Die Regeln für **Template-Abarbeitung** sind recht kompliziert, daher empfiehlt es sich anfangs folgendes zu tun:

- Definiere ein Template für den Wurzelknoten
- Steuere aus diesem Template die weitere Abarbeitung

Beispiel: das Template matcht den Wurzelknoten und gibt den Wert des Attributs aus.

```xml
<A x="25">
  <B>Hallo!</B>
</A>
```
XML

```xsl
<xsl:template match="/A">
   <xsl:value-of select="@x"/>
   In Element B steht: </xsl:value-of select="B"/>
</xsl:template>
```
XSL

Ausgabe:

```
25
In Element B steht: Hallo!
```

Template Matching

Matcht ein Template für einen Knoten, so:

- Wird zuerst das Template ausgeführt
- Für den durch diesen Knoten aufgespannten Teilbaum des Dokuments werden standardmäßig **keine** weiteren Templates angewendet!
  (Ausnahme: es wird explizit aus diesem Template apply-templates aufgerufen

Beispiel: Der Inhalt von D wird nicht ausgegeben, da das Template für B matcht und keine weitere Abarbeitung des Teilbaumes von B gemacht wird

Input

```xml
<A>
  <B>
    <D>Hallo</D>
  </B>
  <C>Welt!</C>
</A>
```

XML

Stylesheet:

```xsl
<xsl:template match="B">
   Element B hat gematcht!
</xsl:template>
<xsl:template match="D">
   Element D hat gematcht!
</xsl:template>
```

XSL

Ausgabe

```
Element B hat gematcht! Welt!
```

Template Matching: apply-templates

Soll also der Teilbaum eines matchenden Elements weiterverarbeitet werden, muss explizit **apply-templates** aufgerufen werden. Im Beispiel zuvor:

Input

```xml
<A>
  <B>
    <D>Hallo</D>
  </B>
  <C>Welt!</C>
</A>
```
XML

Stylesheet:

```xsl
<xsl:template match="B">
   Element B hat gematcht!
   <xsl:apply-templates/>
</xsl:template>
<xsl:template match="D">
   <xsl:value-of select="."/>
</xsl:template>
```
XSL

Ausgabe

```
Element B hat gematcht! Hallo Welt!
```

Bei apply-templates kann man im optionalen Attribut `select` auch noch einen XPath angeben, der einen bestimmten Ausschnitt des Teilbaumes wählt. (Default ist die Auswahl der Kindknoten)

```xsl
<xsl:template match="B">
   Element B hat gematcht!
   <xsl:apply-templates select="D"/>
</xsl:template>
...
```
XSL

Template Matching: apply-templates

Soll also der Teilbaum eines matchenden Elements weiterverarbeitet werden, muss explizit **apply-templates** aufgerufen werden. Im Beispiel zuvor:

Input

```xml
<A>
  <B>
    <D>Hallo</D>
  </B>
  <C>Welt!</C>
</A>
```
XML

Stylesheet:

```xsl
<xsl:template match="B">
   Element B hat gematcht!
   <xsl:apply-templates/>
</xsl:template>
<xsl:template match="D">
   <xsl:value-of select="."/>
</xsl:template>
```
XSL

Ausgabe

```
Element B hat gematcht! Hallo Welt!
```

Bei apply-templates kann man im optionalen Attribut `select` auch noch einen XPath angeben, der einen bestimmten Ausschnitt des Teilbaumes wählt. (Default ist die Auswahl der Kindknoten)

```xsl
<xsl:template match="B">
   Element B hat gematcht!
   <xsl:apply-templates select="D"/>
</xsl:template>
...
```
XSL

Template Matching: Modes

Bei Templates mit gleichem `match` können über das Attribut **`mode`** unterschiedliche Verarbeitungsmodi angedeutet werden

Input:

```xml
<A>
  <B>
    <D>Hallo</D>
  </B>
  <C>Welt!</C>
</A>
```

XML

Stylesheet

```xsl
<xsl:template match="B">
   Element B hat gematcht!
   <xsl:apply-templates/>
   <xsl:apply-templates mode="anders"/>
</xsl:template>
<xsl:template match="D">
   <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="D" mode="anders">
   D mal anders
</xsl:template>
```

XSL

Ausgabe

```
Element B hat gematcht! Hallo D mal anders Welt!
```

Template Matching: Prioritäten

Es kann passieren, dass mehrere match Ausdrücke auf einen bestimmten Knoten matchen

Man kann aber über das Attribut **`priority`** eine Dezimalzahl angeben, welche die Priorität eines Template steuert

Sind keine Prioritäten angegeben, gibt es (komplizierte) Regeln für automatische Vergabe von Prioritäten, die im Bereich zwischen -0,5 und 0,5 liegen.

Vergibt man also händisch Priorität 1, so ist diese höher als jede automatisch vergebene Priorität.

```
<xsl:template match="*">
   STERN
</xsl:template>
<xsl:template match="person">
   PERSON
</xsl:template>
```

Hier matcht das zweite Template, weil es spezifischer ist (es erhält Priorität 0, während das erste Priorität -0,25 erhält)

```
<xsl:template match="*" priority="1">
   STERN
</xsl:template>
<xsl:template match="person">
   PERSON
</xsl:template>
```

Hier matcht das erste Template, weil es explizit Priorität 1 hat (das zweite erhält weiterhin automatisch Priorität 0)

Aufrufen von Templates

Man kann ein Template mit **call-template** explizit aufrufen (ohne für dieses Template ein match anzugeben)

Der Kontextknoten des aufrufenden Template wird dabei an das aufgerufene Template "weitergereicht"

Input:

```xml
<personen>
  <person>
    <name>Michael Derntl</name>
    <email>michael.derntl@univie.ac.at</email>
  </person>
</personen>
```

XML

Stylesheet:

```xsl
<xsl:template match="person">
   <xsl:call-template name="ausgabe"/>
</xsl:template>
<xsl:template name="ausgabe">
   <xsl:value-of select="name"/> (<xsl:value-of select="email"/>)
</xsl:template>
```

XSL

Ausgabe:

```
Michael Derntl (michael.derntl@univie.ac.at)
```

Erzeugen von Output

Beim Erzeugen von Output kann man folgende XSL Tags verwenden:

- Text mit **xsl:text**
- Elemente mit **xsl:element**
- Attribute mit **xsl:attribute**
- Kommentare mit **xsl:comment**

xsl:text

Statt einfach Text in den Output zu schreiben, kann man **xsl:text** verwenden

Folgendes Beispiel erzeugt die gleiche Ausgabe wie vorher, verwendet aber xsl:text um Text in den Output zu schreiben

Input:

```xml
<A>
  <B>
    <D>Hallo</D>
  </B>
  <C>Welt!</C>
</A>
```

XML

Stylesheet:

```xsl
<xsl:template match="B">
   <xsl:text>Element B hat gematcht!</xsl:text>
   <xsl:apply-templates/>
   <xsl:apply-templates mode="anders"/>
</xsl:template>
<xsl:template match="D">
   <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="D" mode="anders">
   <xsl:text>D mal anders</xsl:text>
</xsl:template>
```

XSL

xsl:element

Um ein XML Element im Outputdokument zu erzeugen, verwende **xsl:element**

Input:

```xml
<A>
  <B>
    <D>Hallo</D>
  </B>
  <C>Welt!</C>
</A>
```

XML

Stylesheet:

```xsl
<xsl:template match="/">
   <xsl:element name="p">
      <xsl:value-of select="//D"/>
   </xsl:element>
</xsl:template>
```

XSL

Ausgabe: `<p>Hallo</p>`

Einfügen von Attributen in Elemente des Outputdokuments geht u.a. über **xsl:attribute**

Input:

```xml
<personen>
  <person>
    <name>Michael Derntl</name>
    <email>michael.derntl@univie.ac.at</email>
  </person>
</personen>
```

XML

Stylesheet

```xsl
<xsl:template match="person">
   <xsl:element name="a">
      <xsl:attribute name="href">
         mailto:<xsl:value-of select="email"/>
      </xsl:attribute>
      <xsl:value-of select="name"/>
   </xsl:element>
</xsl:template>
```

XSL

Ausgabe:

```
<a href="mailto:michael.derntl@univie.ac.at">Michael
Derntl</a>
```

Ein Ausdruck in geschwungenen Klammern { } wird bei der Verarbeitung des Stylesheet bzw. des Template durch den Wert des Ausdrucks ersetzt

Gilt nur wenn solche Ausdrücke in Attributwerten verwendet werden!

Für das Beispiel von der vorigen Folie könnte man auch folgendes Stylesheet nehmen (produziert den gleichen Output):

```xsl
<xsl:template match="person">
   <a href="mailto:{email}">
      <xsl:value-of select="name"/>
   </a>
</xsl:template>
```

xsl:for-each

Mit **xsl:for-each** kann man eine Menge von Knoten (durch einen XPath identifiziert) ähnlich einer for-Schleife durchwandern

Innerhalb des for-each ist der Kontextknoten der jeweilige durch den `select` Ausdruck identifizierte Knoten

Beispiel: Alle Bücher ausgeben, deren Preis kleiner 10 ist:

Input:

```xml
<buchhandel>
  <buch preis="25.90">Bibel</buch>
  <buch preis="9.90">Illuminati</buch>
  <buch preis="3.90">Duck Tales</buch>
</buchhandel>
```

XML

Stylesheet:

```xml
<xsl:template match="/">
   Billige Buecher:
   <xsl:for-each select="buchhandel/buch[@preis &lt; 10]">
      <xsl:value-of select="."/>
      <xsl:text>, </xsl:text>
   </xsl:for-each>
</xsl:template>
```

Für die Operatoren **<** und **>** muss man hier die entsprechenden Entities referenzieren!

```
Billige Buecher: Illuminati, Duck Tales,
```

xsl:if

Es können auch Bedingungen mit **xsl:if** für einen bestimmten Abschnitt im Template definiert werden

Für das Beispiel vorher: Auch nach dem letzten Buch wird ein Beistrich ausgegeben. Wir wollen das verhindern:

Stylesheet:

```xsl
<xsl:template match="/">
   Billige Buecher:
   <xsl:for-each select="buchhandel/buch[@preis &lt; 10]">
      <xsl:value-of select="."/>
      <xsl:if test="position()!=last()">
         <xsl:text>, </xsl:text>
      </xsl:if>
   </xsl:for-each>
</xsl:template>
```

XSL

Ausgabe

```
Billige Buecher: Illuminati, Duck Tales
```

(Anmerkung: Es gibt kein xsl:else oder xsl:elseif !)

xsl:choose

Auswahl aus mehreren Möglichkeiten mit **xsl:choose**, **xsl:when** und **xsl:otherwise** (analog switch in anderen Sprachen)

Es sind mehrere xsl:when möglich, aber nur *ein* xsl:otherwise (wird ausgeführt, falls keines der xsl:when zutrifft)

Stylesheet

```xsl
<xsl:template match="/">
   Billige Buecher:
   <xsl:for-each select="buchhandel/buch[@preis &lt; 10]">
      <xsl:choose>
         <xsl:when test="position()=last()">
            <xsl:text>und last but not least: </xsl:text>
            <xsl:value-of select="."/>
         </xsl:when>
         <xsl:otherwise>
            <xsl:value-of select="."/><xsl:text>, </xsl:text>
         </xsl:otherwise>
      </xsl:choose>
   </xsl:for-each>
</xsl:template>
```

XSL

Ausgabe:

```
Billige Buecher: Illuminati, und last but not least: Duck Tales
```

xsl:sort

Innerhalb von xsl:for-each können Teile des Inputdokuments sortiert werden durch einen XPath Ausdruck, der die Werte liefert, nach denen sortiert werden soll

Es können mehrere **xsl:sort** hintereinander gestellt werden (gibt mehrere Sortierkriterien nach Priorität an)

Beispiel: Wir wollen die Bücher alphabetisch sortieren vor der Ausgabe

Stylesheet

```
<xsl:template match="/">
   Billige Buecher:
   <xsl:for-each select="buchhandel/buch[@preis &lt; 10]">
      <xsl:sort select="." order="ascending"/>
      <xsl:value-of select="."/>
      <xsl:if test="position()!=last()">
         <xsl:text>, </xsl:text>
      </xsl:if>
   </xsl:for-each>
</xsl:template>
```

XSL

order ist *ascending* (aufsteigend) oder *descending* (absteigend)

```
Billige Buecher: Duck Tales, Illuminati
```

Variable

Es können Variable definiert werden, die später (woanders) im Stylesheet verwendet werden können

Variable können global (=ausserhalb von Templates) definiert werden oder innerhalb von Templates

- Globale Variable können in allen verwendet werden Templates
- Lokale Variable nur innerhalb des Template wo sie definiert wurden

Referenzieren der Variable über **$name**

Beispiel: Bei jedem Buch ausgeben an welcher Position im Dokument es ist (relativ zur Gesamtanzahl der Bücher)

Stylesheet:

```xsl
<xsl:variable name="anzahl" select="count(//buch)"/>

<xsl:template match="/buchhandel/buch">
  <xsl:value-of select="."/>
  <xsl:text> (</xsl:text>
  <xsl:value-of select="position()"/>
  <xsl:text>/</xsl:text>
  <xsl:value-of select="$anzahl"/>
  <xsl:text>), </xsl:text>
</xsl:template>
```

XSL

Globale Variable ausserhalb eines Template

Ausgabe:

```
Bibel (1/3), Illuminati (2/3), Duck Tales (3/3),
```

Parameters (1)

Es können Parameter definiert werden, die später (woanders) im Stylesheet verwendet werden können, bzw. die man an templates übergeben kann.

Parameter können global (=ausserhalb von Templates) definiert werden oder beim aufruf von Templates.

- Globale Variable können in allen verwendet werden Templates
- Lokale Variable nur innerhalb des Template wo sie definiert wurden

Referenzieren der Variable über **$name**

Beispiel: Berechnen einer Summe

Stylesheet:

```
<xsl:template match="/">
  <xsl:call-template name = "print">
    <xsl:with-param name="A”>11</xsl:with-param>
    <xsl:with-param name="B">33</xsl:with-param>
  </xsl:call-template>
</xsl:template>
<xsl:template name="print">
  <xsl:param name="A"/>
  <xsl:param name="B">111</xsl:param>
  <xsl:value-of select="$A"/>
  <xsl:text> + </xsl:text>
  <xsl:value-of select="$B"/>
  <xsl:text> = </xsl:text>
  <xsl:value-of select="$A+$B"/>
</xsl:template>
```

Ausgabe:

```
11 + 33 = 44
```

Parameters (2)

Parameters können auch von aussen (z.B. java) übergeben werden

- Globalen “param” definieren und eventuell vorbelegen
- Referenzieren des Parameters über $name wie gehabt

Beispiel: nur Bücher mit einem bestimmten Preis ausgeben

Input:

```
<buchhandel>
  <buch preis="25.90">Bibel</buch>
  <buch preis="9.90">Illuminati</buch>
  <buch preis="3.90">Duck Tales</buch>
</buchhandel>
```

Java:

```
StreamSource xslsource = new StreamSource(new File(argv[0]));
TransformerFactory xformFactory =  TransformerFactory.newInstance();
Transformer transformer =     xformFactory.newTransformer(xslsource);
transformer.setParameter(“preis”,”10”);
```

Stylesheet:

```
<xsl:param name="preis" select=”3”/>

<xsl:template match="/">
   Buecher:
   <xsl:for-each select="buchhandel/buch[@preis &lt; $preis]">
      <xsl:value-of select="."/>
      <xsl:if test="position()!=last()">
         <xsl:text>, </xsl:text>
      </xsl:if>
   </xsl:for-each>
</xsl:template>
```

Globale Variable ausserhalb eines Template

Weitere Infos zu XSLT:

W3C Spezifikation für alle, die es ganz genau wissen wollen:
http://www.w3.org/TR/xslt

Tutorial auf W3Schools:
http://www.w3schools.com/xsl

Tutorial auf zvon.org:
http://zvon.org/xxl/XSLTutorial/Output