

Letzte Woche

Themen der letzten Vorlesung:

- Beispiele SURflets
- XML
- Elemente, Tags und Attribute
- Document Typ Deklaration DTD
- DTD: Elemente und Attribute

Entities beschreiben (1)

Es gibt verschiedene Sorten Entities. Generell: Entities sind Platzhalter, die vom Parser durch vordefinierten Text ersetzt werden.

Parsed Internal General Entity

- Definition mit: `<!ENTITY name "text">`
- Wirkung: Ersetze die Vorkommen von `&name`; innerhalb von `#PCDATA` in der XML-Datei durch *text*

Parsed External General Entity

- Funktioniert wie Parsed Internal Entity, der Text befindet sich jedoch außerhalb der DTD.
- Definition mit: `<!ENTITY name SYSTEM URI>`

Entities beschreiben (2)

Internal Parameter-Entity

- Definition mit: `<!ENTITY % name "string">`
- Dürfen nur innerhalb der DTD verwendet werden
- Wirkung: Ersetze die Vorkommen von %*name*; in der DTD durch *string*
- Beispiel: Entity mit regulären Ausdruck von Unterelementen

External Parameter-Entity

- Wie Internal Parameter-Entity, doch Ersetzung außerhalb der DTD definiert
- Definition mit: `<!ENTITY % name SYSTEM URI>`

Beispiel xmlrpc-req.xml

```
<?xml version="1.0"?>
<!DOCTYPE methodCall SYSTEM "xmlrpc.dtd">

<methodCall>
  <methodName>multiply</methodName>
  <params>
    <param>
      <value><int>6</int></value>
    </param>
    <param>
      <value><int>7</int></value>
    </param>
    <param>
      <value><boolean>&>true;</boolean></value>
    </param>
  </params>
</methodCall>
```

Beispiel xmlrpc.dtd

```
<!ELEMENT int (#PCDATA)>
<!ELEMENT boolean (#PCDATA)>
<!ELEMENT string (#PCDATA)>
<!ELEMENT double (#PCDATA)>

<!ENTITY % basic-value "int|boolean|string|double">
<!ELEMENT array (%basic-value;)>
<!ELEMENT value (%basic-value;|array)>

<!ENTITY true "#t">
<!ENTITY false "#f">

<!ELEMENT param (value)>
<!ELEMENT params (param*)>

<!ELEMENT methodName (#PCDATA)>
<!ELEMENT methodCall (methodName, params?)>

<!ELEMENT fault (value)>
<!ELEMENT methodResponse (params|fault)>
```

XML-Dokumente mit `xmllint` überprüfen

DTDs aufteilen

Idee: Komplexe DTDs in mehrere wiederverwendbare DTDs zerlegen

- Mehrere DTDs beschreiben ein XML-Dokument
- Mögliches Problem: Überschneidungen bei den Namen von Elementen

Lösung: Müssen die Sichtbarkeit von Namen steuern können

Weiterer XML-Standard: XML-Namespaces

XML-Namespace

Ein neuer Namespace wird durch das spezielle Attribut `xmlns` deklariert:

```
<tag xmlns:nsprefix="URI"> ... </tag>
```

- `URI` ist der eindeutige Name des Namespaces
- `nsprefix` wird verwendet, um einen *qualifizierten Namen* zu bilden (optional)

Beispiel:

```
<x xmlns:edi='http://ecommerce.org/schema'>  
  <edi:price units='Euro'>15.60</edi:price>  
</x>
```

Hier bezeichnet `edi:price` das Element `price` aus dem Namespace `http://ecommerce.org/schema`

Beispiel: Mehrere Namespaces

Zwei Namensräume gleichzeitig verwenden:

```
<?xml version="1.0"?>
<buchung xmlns:flug="http://localhost/XML/flug"
          xmlns:kunde="http://localhost/XML/kunden">
  <flug:nummer>110</flug:nummer>
  <flug:from>Frankfurt</flug:from>
  <flug:to>Tokio</flug:to>
  <kunde:nummer>4711</kunde:nummer>
</buchung>
```

Beispiel: "Dateninseln"

"Inseln" innerhalb einer XML-Datei:

```
<?xml version="1.0"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>A Math Example</title>
  </head>
  <body>
    <p>The following is MathML markup:</p>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply> <log/>
        <logbase>
          <cn> 3 </cn>
        </logbase>
        <ci> x </ci>
      </apply>
    </math>
  </body>
</html>
```

XML im Browser

XML-Dateien enthalten nur Daten, also keinerlei Informationen für die Darstellung.

Lösung: XML-Datei in (X)HTML umwandeln

Eine mögliche Lösung:

- Definiere die Regeln für die Umwandlung in einer bestimmten Sprache: *eXtensible Stylesheet Language Transformations (XSLT)*
- Die eigentliche Umwandlung führt ein *XSLT-Prozessor* durch (z. B. Browser)

Beispiel XSLT

Ein XML-Dokument bindet ein *Stylesheet* ein:

```
<?xml-stylesheet href="name.xsl" type="text/xsl"?>
```

Das Stylesheet kann auch im XML-Dokument eingebettet werden

Baumrepräsentation von XML

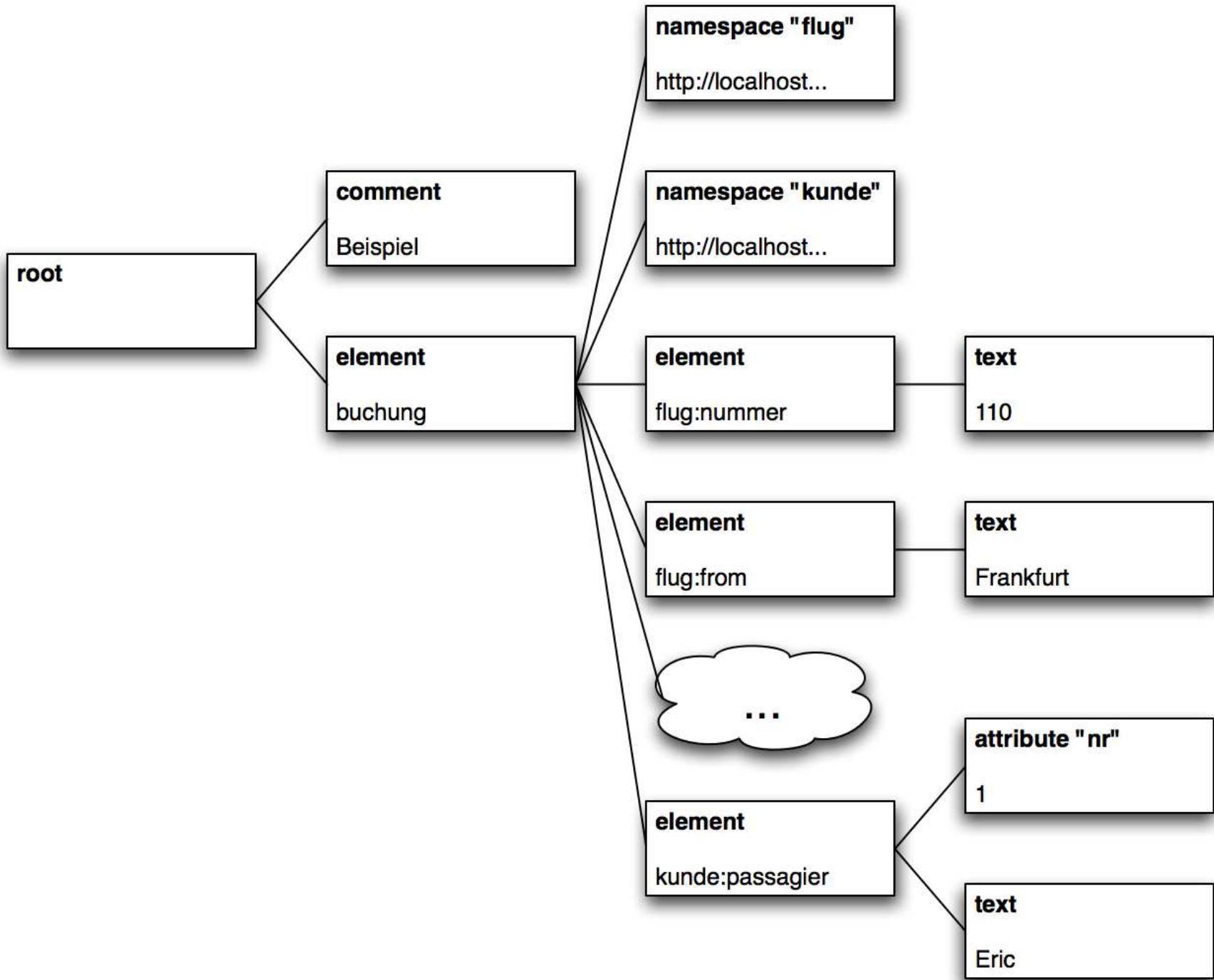
Ein XML-Dokument wird als Baum repräsentiert. XSLT definiert Operationen auf diesem Baum.

Bestandteile eines Baumes:

- Wurzelknoten
- Elementknoten
- Textknoten
- Attributknoten
- Namensraumknoten
- Verarbeitungsanweisungsknoten
- Kommentarknoten

Beispiel: Baumrepräsentation

```
<?xml version="1.0"?>
<!-- Beispiel -->
<buchung xmlns:flug="http://localhost/XML/flug"
          xmlns:kunde="http://localhost/XML/kunden">
  <flug:nummer>110</flug:nummer>
  <flug:from>Frankfurt</flug:from>
  <flug:to>Tokio</flug:to>
  <kunde:nummer>4711</kunde:nummer>
  <kunde:passagier nr="1">Eric</kunde:passagier>
</buchung>
```



Teilbäume und Knoten auswählen

Transformationen werden in der Regel auf Teilbäumen oder Mengen von Knoten ausgeführt

Wie wählt man diese Knoten aus?

XPath beschreibt Verweise auf Teile eines XML-Dokumentes

- XSLT verwendet eine Teilmenge von XPath
- Eigene Spezifikation
- Auch für einfache Berechnungen geeignet
- XPath-Ausdrücke können in *XQuery* verwendet werden

XPath: Kontext und Achse

Kontext

Die Auswertung eines Ausdrucks erfolgt stets relativ zu einem Kontext, (u. a.) bestehend aus:

- Momentaner Knoten
- Momentane Knotenmenge

Achse

Die Achse bestimmt welches Element des momentanen Knotens auf ein Vergleichsmuster angewendet wird.

- Standard ist die **child**-Achse, also die direkt unterhalb angeordneten Knoten
- Andere Achsen: **parent**, **self** und **attribute**
- Insgesamt werden 13 Achsen unterschieden

XPath-Ausdrücke (1)

Wichtige XPath-Ausdrücke:

<i>name</i>	alle Kind-Knoten des Typs <i>name</i>
*	alle Kind-Knoten
<i>e1/e2</i>	alle Elemente des Typs <i>e2</i> , die ein Element des Typs <i>e1</i> als Elternknoten haben
<i>e1//e2</i>	alle Elemente des Typs <i>e2</i> , die ein Element des Typs <i>e1</i> als Vorgänger haben
/	der Wurzelknoten
<i>name[expr]</i>	alle Kind-Elemente des Typs <i>name</i> für die <i>expr</i> zu wahr evaluiert
@ <i>name</i>	das Attribut <i>name</i> des Knotens
<i>s1 s2</i>	alle Elemente des Typs <i>s1</i> oder <i>s2</i>

XPath-Ausdrücke (2)

- Arithmetische Ausdrücke und Vergleichsoperatoren
`+ - * div mod and or = != > < >= <=`
- Funktionen auf Knotenlisten
`last() position() count()`
- Für Pfade im Baum
`• •`
- Operation auf bestimmter Achse: `name::expr`
Z. B.: `attribute::name`

XSLT Stylesheets

XSLT Stylesheets sind ein XML-Dokument mit dem Namensraum `http://www.w3.org/1999/XSL/Transform`:

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

Außerdem: Wurzel-Element `stylesheet` mit Attribut `version`

Die wichtigsten Elemente innerhalb eines Stylesheets:

- `template` Eine Regel, um Elemente zu transformieren
- `output` Legt fest, wie das Resultat der Transformation aussieht:
`xml`, `html`, `text`

Templates

Ein Template definieren:

```
<xsl:template match = pattern  
              name = qname  
              priority = number>
```

Transformation

```
</xsl:template>
```

- *pattern*: XPath-Ausdruck, der bestimmt welche Elemente transformiert werden
- *name*: Name für die Regel
- *priority*: Reihenfolge der Abarbeitung im Konfliktfall

Ablauf einer XML-Transformation

Laufe die Baumrepräsentation des Dokuments bei der Wurzel beginnend ab:

- Erzeuge Liste, bestehend aus dem Wurzelknoten
- Verarbeitet eine Liste aus Knoten, die die verarbeiteten Unterlemente darstellen
- Überprüft die Regeln für jedes Listenelement, wählt Regel mit höchster Priorität aus. Die Regel wird mit dem gewählten Knoten instanziiert

Das Template wählt in der Regel weitere Knoten aus, dort wird der Prozess fortgesetzt.

XSLT-Anweisungen (1)

Innerhalb eines Templates können XSLT-Anweisungen verwendet werden. Auswahl wichtige Anweisungen:

- **xsl:apply-templates** Das Attribut `select` (XPath-Ausdruck) wählt Kindknoten zur weiteren Verarbeitung aus
- **xsl:attribute** Fügt ein Attribut zum Ergebnisknoten hinzu
- **xsl:value-of** Erzeugt einen Textknoten aus der Stringrepräsentation eines Knotens
- **xsl:text** Erzeugt einen Textknoten aus dem Inhalt des Elements
- **xsl:for-each** Wendet den Inhalt des Elements auf jeden Knoten an, den das Attribut `select` auswählt

XSLT-Anweisung (2)

- **xsl:if** Führt Template im Inhalt an, wenn der Ausdruck im Attribut `test` wahr ergibt
- **xsl:choose** Unterscheidet durch **xsl:when**-Elemente gegebene Fälle, analog zu **xsl:if**. Falls kein Fall wahr ist, wird evt. vorhandenes **xsl:otherwise** angewendet
- **xsl:sort** Sortiert Knotenliste nach dem im Attribut `select` angegebenen Schlüssel
- **xsl:message** Gibt den Inhalt auf der Konsole aus

Beispiel: pfi.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE veranstaltung SYSTEM "veranstaltung.dtd">
<?xml-stylesheet href="kommvv.xsl" type="text/xsl"?>
<veranstaltung jahr="2005" art="Vorlesung"
  titel="Programmieren für das Internet">
  <dozent>
    <name>Prof. Dr. H. Klaeren</name>
  </dozent>
  <dozent>
    <name>Eric Knauel</name>
  </dozent>
  <dozent>
    <name>Mandeep Singh</name>
  </dozent>
  <zeit_änderung_nach_release="ja">
    <wochentag>Dienstag</wochentag>
    <uhrzeit>15:15-17:00</uhrzeit>
  </zeit>
  <raum>Sand 6/7, großer Hörsaal</raum>
</veranstaltung>
```

Beispiel: kommvv-simple.xsl (1)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="html" encoding="ISO-8859-1"/>
  <xsl:template match="veranstaltung">
    <html>
      <body>
        <title><xsl:value-of select="@titel"/></title>
        <h1><xsl:value-of select="@titel"/></h1>
        <p>Dozenten:
          <xsl:apply-templates select="dozent"/>
        </p>
        <xsl:apply-templates select="zeit"/>
        <xsl:apply-templates select="raum"/>
        <hr/>
        <p>Alle Angaben ohne Gewähr</p>
      </body>
    </html>
  </xsl:template>
```

Beispiel: kommvv-simple.xsl (2)

```
<xsl:template match="dozent">  
  <xsl:apply-templates select="name"/>  
</xsl:template>
```

```
<xsl:template match="zeit">  
  <p> Uhrzeit: <em> <xsl:apply-templates/></em></p>  
</xsl:template>
```

```
<xsl:template match="raum">  
  <p>Raum: <em> <xsl:apply-templates/> </em></p>  
</xsl:template>
```

```
</xsl:stylesheet>
```

Beispiel: kommvv.xsl (1)

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

  <xsl:output method="html" encoding="ISO-8859-1"/>
  <xsl:template match="veranstaltung">
    <html>
      <body>
        <title><xsl:value-of select="@titel"/></title>
        <h1><xsl:value-of select="@titel"/></h1>
        <p>Dozenten:</p>
        <ul>
          <xsl:for-each select="dozent">
            <li>
              <xsl:apply-templates select="self::node()"/>
            </li>
          </xsl:for-each>
        </ul>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Beispiel: kommvv.xsl (2)

```
<xsl:apply-templates select="zeit"/>
<xsl:apply-templates select="raum"/>
<hr/>
<p>Alle Angaben ohne Gewähr</p>
</body>
</html>
</xsl:template>
```

Beispiel: kommvv.xsl (3)

```
<xsl:template match="dozent[position()&lt;last()]">
  <xsl:apply-templates select="name"/>,
</xsl:template>
<xsl:template match="dozent">
  <xsl:apply-templates select="name"/>
</xsl:template>
```

Beispiel: kommvv.xsl (4)

```
<xsl:template match="zeit">
  <p>
    Uhrzeit: <em>
      <font>
        <xsl:choose>
          <xsl:when test="@änderung_nach_release='ja'">
            <xsl:attribute name="color">#FF0000</xsl:attribute>
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name="color">#000000</xsl:attribute>
          </xsl:otherwise>
        </xsl:choose>
        <xsl:apply-templates select="node()" />
      </font>
    </em>
  </p>
</xsl:template>
```

Beispiel: kommvv.xsl (5)

```
<xsl:template match="raum">  
  <p>Raum: <em> <xsl:apply-templates/> </em></p>  
</xsl:template>  
</xsl:stylesheet>
```