XML Kurs

Markus Degen / Prof. Peter Thiemann

12. Oktober 2005

Was ist XML?

- eXtensible Markup Language
- beschreibt Daten
- Ähnlichkeit mit HTML
- definierbare logische Struktur:
  - markierte, sortierte Bäume
  - keine Vordefinirten Tags
- mächtige Hyperlinkmöglichkeiten (XLink)
- Transformationssprachen (XPath, XSL)

Unterschied zu HTML

- XML beschreibt Daten/Inhalte
- HTML stellt Daten/Inhalte dar

Beispiel: Wohlgeformtes XML Dokument

```
<?xml version="1.0"?>
<Eltern>
  <Kind>
    Hier kommt der Inhalt.
  </Kind>
  <Leer warum="darum" />
</Eltern>
```

XML Deklaration: `<?xml version="1.0"?>`

Elemente: `Eltern`, `Kind`, `Leer`

Öffnende Tags: `<Eltern>`, `<Kind>`

Schließende Tags: `</Kind>`, `</Eltern>`

Leeres Element: `<Leer />`

Attribut: `warum="darum"`
(im öffnenden Tag, bzw. im Tag des leeren Elements)

Wohlgeformtheit (well-formedness)

Wohlgeformt heißt

- XML Deklaration vorhanden
- sämtliche Elemente haben öffnendes und schließendes Tag (oder `<Leer />`, falls kein Inhalt)
- korrekte Schachtelung der Tags!
- genau ein Wurzelelement vorhanden
- jeder Attributname höchstens einmal pro Element

$\Rightarrow$ Dokument ist durch einen Baum repräsentierbar

Element Groß-/Kleinschreibung beachten!

Attribute müssen immer in Anführungsstrichen stehen!

falsch `<note date=12/11/2002>`

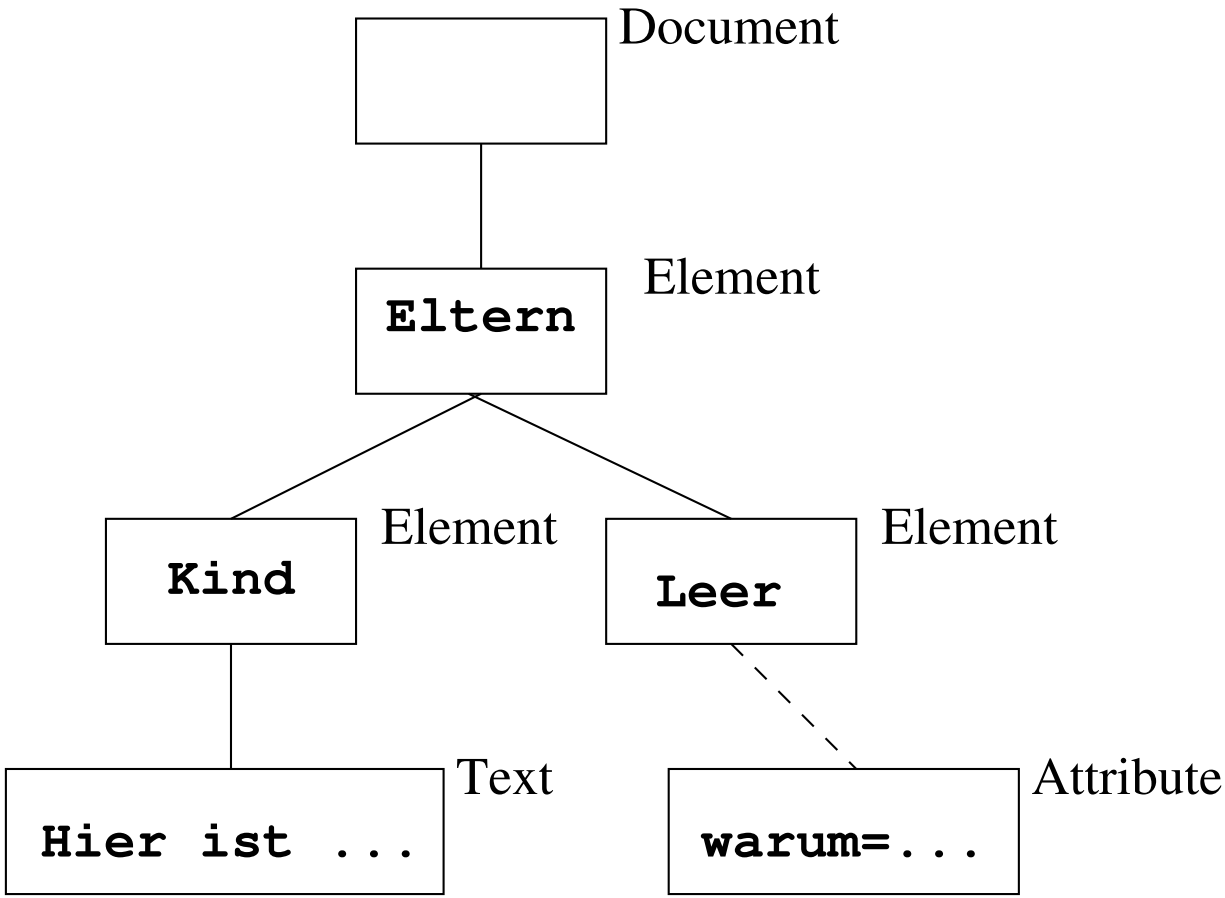
richtig `<note date="12/11/2002">`

Attributnamen müssen in einem Element eindeutig sein

Leerzeichen werden nicht entfernt!

Kommentare `<!-- Kommentar -->`

Baumdarstellung des Dokuments



Knotentypen in XML Dokumenten

(XPath Terminologie)

Node	übergeordneter Knotentyp
Text	enthält Text, keine Kinder
Element	enthält andere Knoten als Kinder (insbesondere Text, Element), sowie Attribute (sind nicht Kinder). Kinder von Elementen sind geordnet
Attribut	Name/Wert-Paar anhängend an einem Elementknoten
Comment	zählt weder zum Inhalt noch zur Struktur, keine Kinder
Processing Instruction	Anweisung an XML-verarbeitendes Programm
Document	Wurzelknoten eines XML Dokuments; enthält genau ein Element, das Wurzelelement

- Knoten eines Dokuments sind durch *Dokumentenordnung* total geordnet
- gegeben durch Baumdurchlauf in preorder von links nach rechts
- entsprechend der Reihenfolge in XML Notation
- Baumterminologie
  - root
  - siblings
  - leaves
  - children
  - parent
  - descendants
  - ancestors

Elementnamen

Elementnamen müssen die folgenden Bedingungen erfüllen

- Namen bestehen aus Buchstaben, Zahlen und Sonderzeichen
- Namen dürfen nicht mit einer Zahl oder einem Satzzeichen beginnen
- Namen dürfen nicht mit den Buchstaben "xml" beginnen
- Namen dürfen keine Leerzeichen enthalten

Gültigkeit eines XML Dokuments

- muss die logische Struktur definieren

→ DTD (Document Type Definition)

- DTD ordnet jedem Elementnamen e einen regulären Ausdruck $R(e)$ über Elementnamen zu
  falls <e><e_1 . . .><e_2 . . .> . . .<e_n . . .></e> in gültigem Dokument vorkommt, so ist $e_1 e_2 \ldots e_n \in L(R(e))$.
  dh die Namen der Kindelemente von e müssen dem Ausdruck $R(e)$ entsprechen
- DTD ordnet jedem Elementnamen eine Menge von getypten Attributen zu
- DTD definiert Abkürzungen (*entities*), die vom XML-Parser expandiert werden

DTD Beispiel

```
<?xml version="1.0"?>
<!DOCTYPE ELTERN [
  <!ELEMENT ELTERN (KIND*)>
  <!ELEMENT KIND (MARKE?,NAME+)>
  <!ELEMENT MARKE EMPTY>
  <!ELEMENT NAME (NACHNAME+,VORNAME+)*>
  <!ELEMENT NACHNAME (#PCDATA)>
  <!ELEMENT VORNAME (#PCDATA)>
  <!ATTLIST MARKE
            NUMMER ID #REQUIRED
            GELISTET CDATA #FIXED "ja"
            TYP (natürlich|adoptiert) "natürlich">
  <!ENTITY STATEMENT "Wohlgeformtes XML">
]>
<ELTERN>
 &STATEMENT;
 <KIND>
  <MARKE NUMMER="1" GELISTET="ja" TYP="natürlich" />
  <NAME>
   <NACHNAME>Flavius</NACHNAME>
   <VORNAME>Secundus</VORNAME>
  </NAME>
 </KIND>
</ELTERN>
```

Kopfzeilen einer XML Datei

Vorspann

- `<?xml version="1.0" encoding="UTF-8"?>`
- Spezifikation einer internen DTD

  ```
  <!DOCTYPE Name [
    Element-, Attribut-, Entitydeklarationen
  ]>
  ```

- Spezifikation einer externen DTD

  ```
  <!DOCTYPE root-element SYSTEM "filename">
  ```

  Beispiel:

  ```
  <!DOCTYPE HTML3
    PUBLIC "-//IETF//DTD HTML Strict//EN//3.0"
           "html-3s.dtd"
           [ ggf. interne Deklarationen ]>
  ```

Aufbau einer DTD

Inhalte der XML Datei aus Sicht der DTD:

- Elemente
- Tags
- Attribute
- Entities
- PCDATA
- CDATA

Elementdeklaration

$$\langle elementdecl \rangle \quad ::= \quad \texttt{<!ELEMENT} \ \langle Name \rangle \ \langle contentspec \rangle \texttt{>}$$
$$\langle contentspec \rangle \quad ::= \quad \texttt{EMPTY} \mid \texttt{ANY} \mid \langle Mixed \rangle \mid \langle children \rangle$$

- Produktion einer kf. Grammatik
- $\langle Mixed \rangle$: `#PCDATA` und Elemente vermischt
- $\langle children \rangle$: regulärer Ausdruck über Elementnamen; Operatoren `,` `|` `?` `+` `*`

Element Beispiele

```
<!ELEMENT br EMPTY>

<!ELEMENT from (#PCDATA)>

<!ELEMENT note (to+,from,header?,(message|#PCDATA))*>
```

Attributdeklaration

$$\langle AttlistDecl \rangle \quad ::= \quad \texttt{<!ATTLIST } \langle Name \rangle \ \langle AttDef \rangle^* \texttt{>}$$
$$\langle AttDef \rangle \quad ::= \quad \langle Name \rangle \ \langle AttType \rangle \ \langle DefaultDecl \rangle$$

$\langle AttType \rangle$ ::=

`CDATA`	String
`ID`	eindeutiger Schlüssel
`IDREF, IDREFS`	Verweis(e) auf Schlüssel
`ENTITY, ENTITIES`	Name(n) von Entities
`NMTOKEN, NMTOKENS`	ein oder mehrere Wörter
$(Nmtoken \ (\mid \ Nmtoken)^*)$	Aufzählungstyp

⟨*DefaultDecl*⟩ ::=

#REQUIRED	erforderlich
| #IMPLIED	nicht erforderlich, kein Default
| "⟨*AttValue*⟩"	Defaultwert
| #FIXED "⟨*AttValue*⟩"	fester Wert

Attribut Beispiele

```
<!ATTLIST payment type CDATA "check">

<!ATTLIST contact fax CDATA #IMPLIED>

<!ATTLIST payment type (check|cash) "cash">
```

Entity Deklaration

$$\langle GEDecl \rangle \quad ::= \quad \texttt{<!ENTITY } \langle Name \rangle \ \langle EntityDef \rangle \texttt{ >}$$
$$\langle EntityDef \rangle \quad ::= \quad \langle EntityValue \rangle \mid (\langle ExternalID \rangle \ \langle NDataDecl \rangle ?)$$

- $\langle GEDecl \rangle$ *general entity*
  Abkürzungen, Dokumentenbausteine (auch von Dateien $\langle ExternalID \rangle$)
  `<!ENTITY dd "Dagobert Duck">`

- Sonderzeichen

Entity References	Character
&lt;	<
&gt;	>
&amp;	&
&quot;	"
&apos;	'

- `PCDATA` *parsed character data*
- `CDATA` *character data*

XML Namespaces

"Modulsystem für XML"
Vermeidung von Namenskonflikten bei Benutzung von

- mehreren Quellen für Elementdeklarationen oder
- mehreren Anwendungen mit dem gleichen Dokument

XML Path Language (XPath)

Ziel von XPath: Spezifikation von Folgen von Knoten in XML-Dokument

- Verwendung in URIs und Attributen
- kompakte Syntax (nicht XML)
- operiert auf logischer Struktur des Dokuments
- einfache Berechnungen auf Ergebnismengen
- weitere Verwendung in anderen XML-Standards: XSLT, XQuery, XPointer, XLink, XForms, ...
- XPath 1.0 $\subseteq$ XPath 2.0

XPath-Sicht von XML

XML-Dokument ist ein Baum mit folgenden Knotenarten

- Wurzel (Dokumentenknoten)
- Element
- Attribut
- Text
- Namespace
- Verarbeitungsanweisungen <?*name daten*?>
- Kommentare

Pfadausdrücke

Ein Pfadausdruck ...

- Folge von Schritten $\langle Step \rangle$ getrennt durch /.
- Absoluter Pfadausdruck beginnt mit /.
- definiert Folge von XML-Knoten.

$$
\begin{array}{lcl}
\langle LocationPath \rangle & ::= & \langle RelativeLocationPath \rangle \\
 & | & \langle AbsoluteLocationPath \rangle \\
\langle AbsoluteLocationPath \rangle & ::= & /\langle RelativeLocationPath \rangle ? \\
 & | & \langle AbbreviatedAbsoluteLocationPath \rangle \\
\langle RelativeLocationPath \rangle & ::= & \langle Step \rangle \\
 & | & \langle RelativeLocationPath \rangle / \langle Step \rangle \\
 & | & \langle AbbreviatedRelativeLocationPath \rangle
\end{array}
$$

Location $\langle Step \rangle$ besteht aus

- Achse (traversierte Beziehung zwischen Knoten)
- Knotentest (Typ und Name)
- optionale Prädikate

$$
\begin{array}{lcl}
\langle Step \rangle & ::= & \langle AxisSpecifier \rangle \ \langle NodeTest \rangle \ \langle Predicate \rangle^* \\
 & | & \langle AbbreviatedStep \rangle \\
\langle AxisSpecifier \rangle & ::= & \langle AxisName \rangle \ :: \\
 & | & \langle AbbreviatedAxisSpecifier \rangle
\end{array}
$$

Syntax:

axisname::nodetest[predicate]

Ergebnis eines Pfadausdrucks: Folge der Knoten (in Dokumentenordnung), die entlang der Achse erreicht werden, den Knotentest erfüllen und sämtliche Prädikate erfüllen

Beispiel: Rezepte

```
child::rcp:recipe[attribute::id='117'] /
child::rcp:ingredient /
attribute::amount
```

```
<rcp:recipe id='116'>
  ...
</rcp:recipe>
<rcp:recipe id='117'>
  ...
  <rcp:ingredient amount='42'>
  </rcp:ingredient>
  ...
  <rcp:ingredient amount='4711'>
  </rcp:ingredient>
  ...
</rcp:recipe>
<rcp:recipe id='118'>
  ...
</rcp:recipe>
```

Beispiel: Rezepte

```
child::rcp:recipe[attribute::id='117'] /
child::rcp:ingredient /
attribute::amount
```

```
<rcp:recipe id='116'>
  ...
</rcp:recipe>
<rcp:recipe id='117'>
  ...
  <rcp:ingredient amount='42'>
  </rcp:ingredient>
  ...
  <rcp:ingredient amount='4711'>
  </rcp:ingredient>
  ...
</rcp:recipe>
<rcp:recipe id='118'>
  ...
</rcp:recipe>
```

Beispiel: Rezepte

```
child::rcp:recipe[attribute::id='117'] /
child::rcp:ingredient /
attribute::amount
```

```
<rcp:recipe id='116'>
  ...
</rcp:recipe>
<rcp:recipe id='117'>
  ...
  <rcp:ingredient amount='42'>
  </rcp:ingredient>
  ...
  <rcp:ingredient amount='4711'>
  </rcp:ingredient>
  ...
</rcp:recipe>
<rcp:recipe id='118'>
  ...
</rcp:recipe>
```

Beispiel: Rezepte

```
child::rcp:recipe[attribute::id='117'] /
child::rcp:ingredient /
attribute::amount
```

```
<rcp:recipe id='116'>
  ...
</rcp:recipe>
<rcp:recipe id='117'>
  ...
  <rcp:ingredient amount='42'>
  </rcp:ingredient>
  ...
  <rcp:ingredient amount='4711'>
  </rcp:ingredient>
  ...
</rcp:recipe>
<rcp:recipe id='118'>
  ...
</rcp:recipe>
```

Alle XPath-Achsen

⟨*AxisName*⟩ ::=

- `child` — `parent`
- `descendant` — `ancestor`
- `following-sibling` — `preceding-sibling`
- `following` — `preceding`
- `attribute`
- `namespace`
- `self`
- `descendant-or-self` — `ancestor-or-self`

Knotentests

⟨*NodeTest*⟩	::=	⟨*NameTest*⟩	
	|	⟨*NodeType*⟩()	
⟨*NameTest*⟩	::=	*	jeder Knoten
	|	⟨*NCName*⟩:*	beliebiges Element in Namespace
	|	⟨*QName*⟩	benanntes Element
⟨*NodeType*⟩	::=	`comment`	true, falls Kommentarknoten
	|	`text`	falls Textknoten
	|	`processing-instruction`	falls Verarbeitungsanweisung
	|	`node`	immer true

Prädikate

$$\langle Predicate \rangle \quad ::= \quad [\langle Expr \rangle]$$

Für jeden durch Achse und $\langle NodeTest \rangle$ selektierten Knoten wird Ausdruck $\langle Expr \rangle$ im Kontext ausgewertet. Ergebnis $\rightarrow$ Boolean. Falls Ergebnis `false`, wird der Knoten verworfen.

Weitere Ausdrücke Boolesche Operationen, arithmetische Operationen, Vergleichsoperationen, Stringoperationen (über Funktionsaufrufe), Operationen auf Knotenmengen

Abkürzungen

In Location ⟨*Step*⟩s gelten folgende Abkürzungen

`child::`		ist optional
`//`	für	`/descendant-or-self::node()/`
`@`	für	`attribute::`
`.`	für	`self::node()`
`..`	für	`parent::node()`

⟨*AbbreviatedAbsoluteLocationPath*⟩ ::= //⟨*RelativeLocationPath*⟩
⟨*AbbreviatedRelativeLocationPath*⟩ ::= ⟨*RelativeLocationPath*⟩//⟨*Step*⟩
⟨*AbbreviatedStep*⟩ ::= . | ..
⟨*AbbreviatedAxisSpecifier*⟩ ::= @?

Beispiele

```
/bookstore/book[1]
/bookstore/book[last()]
/bookstore/book[last()-1]
/bookstore/book[position()<3]
//title[@lang]
//title[@lang='eng']
/bookstore/book[price>35.00]/title
```

Extensible Stylesheet Language (XSL)

Was ist ein Stylesheet?

Spezifikation der Formatierung eines XML-Dokuments.

Wozu dient XSL?

XSLT	Transformation von XML-Dokumenten
XSL-FO	vordefinierte Formatierungskomponenten

Vorgänger:

CSS	(cascaded stylesheets)
DSSSL	(document stylesheet specification language)

Vorgänger: CSS

- Spezifikation des physikalischen Layout zu einer logischen Struktur
- Darstellung von (X)HTML und XML
- Zuordnung: Element im Kontext $\longrightarrow$ Stilmerkmal (Grösse, Font, Farbe, ...)
- Aktuelle Version CSS2.1; bald CSS3

Zuordnung: Dokument → CSS

- Im <head> von XHTML Dokument
- Intern durch <style> Element

```
<head>
<style type="text/css">
p.special {color: green; border: solid red;}
</style>
</head>
```

- Extern durch Verweis auf CSS Datei

```
<head>
  <link rel="stylesheet" href="style.css" type="text/css"/>
</head>
```

CSS Beispiele

`p`	<p> Element
`p.special`	<p> Element mit Attribut `class="special"`
`.special`	Element mit `CLASS="special"`
`ul, p`	<p> oder
`#xy4711`	nur Element mit `id="xy4711"`
`ul p`	<p> unterhalb von
`li > p`	<p> direkt unterhalb von
`a[title]`	<a> mit Attribut `title`
`a[attr = "wert"]`	<a>, bei dem `attr` den Wert `wert` hat
`a[attr ≅ "wert"]`	<a>, bei dem `attr` das Wort `wert` enthält

Beispiel: Anzeige von Visitenkarten

- Visitenkarten liegen in XML Format vor
- Gewünscht: Schönes Anzeigeformat

```
<card xmlns="http://businesscard.org">
  <name>John Doe</name>
  <title>CEO, Widget Inc.</title>
  <email>john.doe@widget.inc</email>
  <phone>(202) 555-1414</phone>
  <logo uri="widget.gif"/>
</card>
```

Visitenkarte mit CSS

- Am Anfang von `card.xml` (Browser: Mozilla)

```
<?xml-stylesheet type="text/css" href="card.css"?>
```

- `card.css`

```
card { background-color: #cccccc; border: none; width: 300;}
name { display: block; font-size: 20pt; margin-left: 0; }
title { display: block; margin-left: 20pt;}
email { display: block; font-family: monospace; margin-left: 20pt;}
phone { display: block; margin-left: 20pt;}
```

- Ergebnis

- Gleiche Reihenfolge
- Information in Attributen?
- Keine neue Strukturen

Visitenkarte mit XSLT

- Am Anfang von `card.xml` (Browser: Mozilla)
  ```
  <?xml-stylesheet type="text/xsl" href="card.xsl"?>
  ```
- Ergebnis

Beispiel: XSLT Formatierung der Visitenkarte

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0"
                xmlns:b="http://businesscard.org"
                xmlns="http://www.w3.org/1999/xhtml" >

  <xsl:output method="html"/>

  <!-- template rules on following slides -->

</xsl:stylesheet>
```

Beispiel: Template Rule Ia

```
<xsl:template match="b:card">
  <html>
    <head>
      <title><xsl:value-of select="b:name"/></title>
    </head>
    <body bgcolor="#FFFFFF">
      <table border="3">
        <tr>
          <td>
            <xsl:apply-templates select="b:name"/> <br/>
            <xsl:apply-templates select="b:title"/> <br/>
            <tt><xsl:apply-templates select="b:email"/></tt> <br/>
            <xsl:if test="b:phone">
              Phone: <xsl:apply-templates select="b:phone"/>
            </xsl:if>
          </td>
```

Beispiel: Template Rule Ib

```
            <td>
              <xsl:if test="b:logo">
                <img src="{b:logo/@uri}"/>
              </xsl:if>
            </td>
          </tr>
        </table>
      </body>
    </html>
  </xsl:template>
```

Beispiel: Template Rule II

```
<xsl:template match="b:name|b:title|b:email|b:phone">
  <xsl:value-of select="."/>
</xsl:template>
```

- Eigentlich überflüssig, da durch Standardregeln abgedeckt

Überblick XSLT

XSLT-Prozessor:

- Dokument `abc.xml` (mit `stylesheet` Direktive auf)
- Stylesheet `abc.xsl`

1. (Baum-) Transformation von `abc.xml`
   spezifiziert durch Menge von Transformationsregeln
   Regel: Pattern $\longrightarrow$ Template *(wie funktionale Sprache)*
2. Formatierung des Ergebnisses
   nach Abschluss der Transformation
   spezieller Namespace für Formatierungsobjekte
   Eigenschaften (*properties*) für Farbe, Größe, Font, usw

template

Das `<xsl:template>` Element erzeugt ein Template

Mit dem Attribut `match` wird das entsprechende xml-Element ausgewählt, für das das Template verwendet wird.

Beispiel: `<xsl:template match="/">`

value-of

Mit `<xsl:value-of>` wird der Inhalt eines Elementes extrahiert.

Beispiel: `<xsl:value-of select="catalog/cd/title"/>`

for-each

Mit `<xsl:for-each>` werden alle Elemente einer Knotenmenge betrachtet.

Beispiel: `<xsl:for-each select="catalog/cd">`

sort

Mit `<xsl:sort>` können die Elemente sortiert ausgegeben werden.

Beispiel: `<xsl:sort select="artist"/>`

if

`<xsl:if>` ermöglicht Bedingungen.

Beispiel: `<xsl:if test="price &gt; 10">`

choose

<xsl:choose> ermöglicht Bedingungen mit mehreren Verzweigungen.

Beispiel:

```
<xsl:choose>
  <xsl:when test="price &gt; 10">
    ...
  </xsl:when>
  <xsl:when test="price &gt; 9 and price &lt;= 10">
    ...
  </xsl:when>
  <xsl:otherwise>
    ...
  </xsl:otherwise>
</xsl:choose>
```

apply-templates

Mit `<xsl:apply-templates>` werden andere Templates für das entsprechende Element aufgerufen.

Beispiel: `<xsl:apply-templates select="title"/>`