Freie Universität Berlin

XSLT: Transformation von XML-Dokumenten

Heutige Vorlesung

Freie Universität Berlin

letzte Woche

- ☑ Welche XML-Parser gibt es?
- ☑ Was sind ihre Vor- und Nachteile?
- ☑ Schema-Übersetzer als Alternative

heutige Vorlesung

- Warum XML-Dokumente transformieren?
- XML-Dokumente mit XSLT transformieren
- XSL-FO zur Erzeugung von druckfähigem Layout

Warum XML transformieren?

Freie Universität Berlin

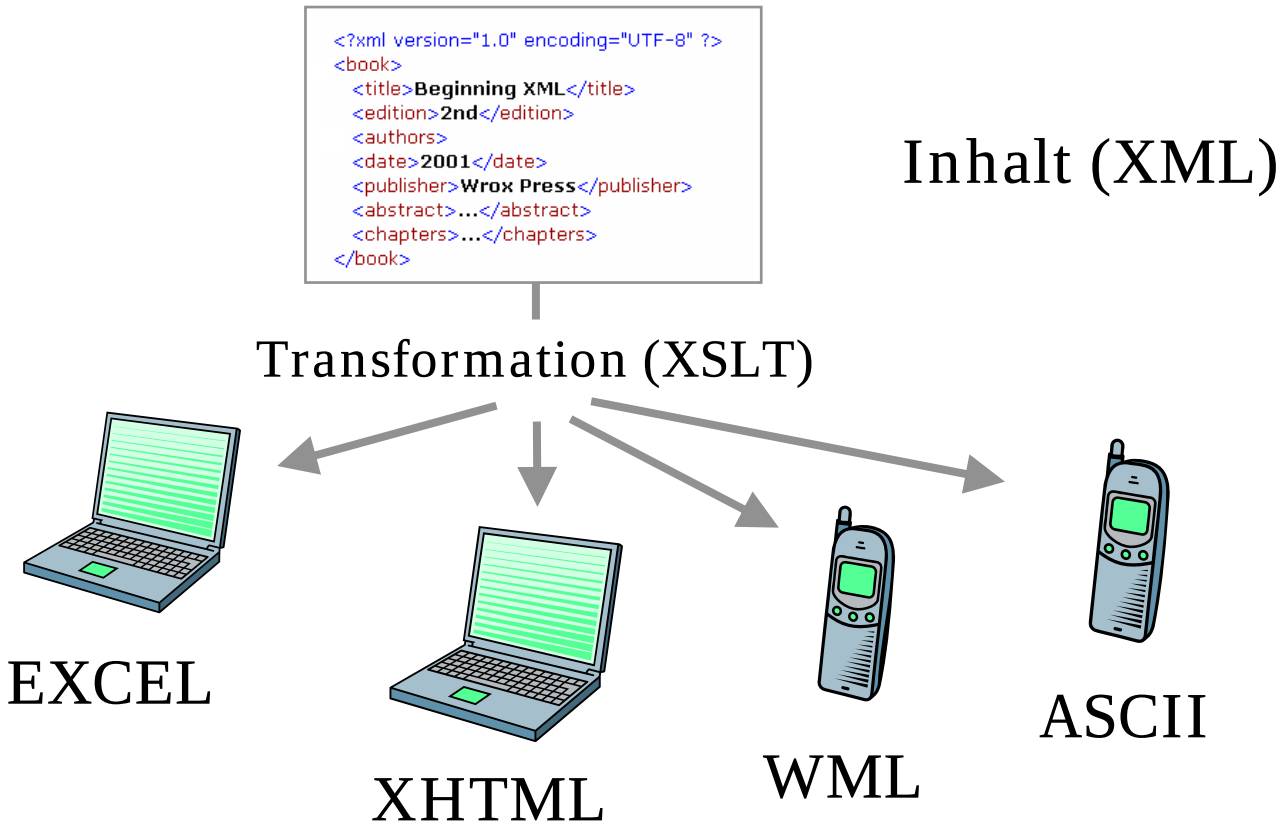
Trennung Inhalt und Präsentation

- XML trennt Inhalt von Präsentation (Layout)
- Für eine entsprechende Darstellung müssen XML-Inhalte transformiert werden:
  - XML-Inhalt ➔ Layout

Inhaltliche Transformationen

- Daten mit XML repräsentiert
- unterschiedliche Sichten (Views) auf XML-Inhalte erfordern Transformationen:
  - XML-Inhalt ➔ XML-Inhalt

XML-Inhalt ➔ Layout

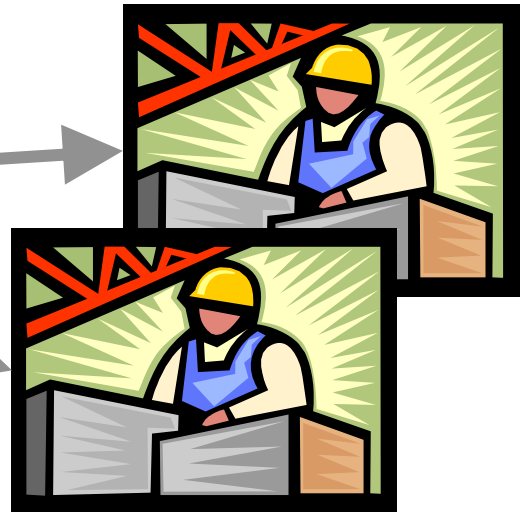


- Multi-Delivery: unterschiedliches Layout von Inhalten
- Beachte: XHTML, WML ⊂ XML

XML-Inhalt ➔ XML-Inhalt

Großhandel

Zulieferer



interner Kundenauftrag

- ~~Name des Verkäufers~~
- Datum
- Produktbezeichnung aus Großhandelskatalog
- Anzahl
- ~~Kunde~~

anpassen

übernehmen

externer Zulieferauftrag

- Datum
- Produktbezeichnung aus Zuliefererkatalog
- Anzahl
- Auftraggeber

XML-Inhalt ➔ XML-Inhalt

Freie Universität Berlin

Kundenauftrag

```
<?xml version="1.0"?>
<order>
 <salesperson>John Doe</salesperson>
 <item>Production-Class Widget</item>
 <quantity>16</quantity>
 <date>
   <month>1</month>
   <day>13</day>
   <year>2000</year>
 </date>
 <customer>Sally Finkelstein</cus
</order>
```

andere Sicht (view)
auf XML-Inhalt

Zulieferauftrag

```
<?xml version="1.0" encoding="UTF-8"?>
<order>
 <customer>Company A</customer>
 <date>2000/1/13</date>
 <item>
   <part-number>E16-25A</part-number>
   <description>Production-Class Widget</description>
   <quantity>16</quantity>
 </item>
</order>
```

Freie Universität Berlin

XSLT

Was ist XSLT?

Freie Universität Berlin

- Programmiersprache zur Transformation von XML-Dokumenten
- erlaubt XML-Dokumente in beliebige Textformate zu Transformieren:

  XML ➔ XML/HTML/XHTML/WML/RTF/ASCII ...
- XSLT-Programme (stylesheets) haben XML-Syntax

  ➔ plattformunabhängig
- W3C-Standard seit 1999

XSLT: Anfragesprache für XML?

Freie Universität Berlin

SQL

- Anfrage = Sicht (View) auf Menge von Relationen
- abgeschlossen: SQL-Anfrage liefert immer eine Relation

XSLT

- Transformation = Sicht (View) auf Menge von XML-Dokumenten

⇨ Anfragesprache für XML

- nicht abgeschlossen: kann beliebige Textformate liefern, nicht nur wohlgeformtes XML

Programmierparadigma

XSLT-Programm (stylesheet)

= Menge von Transformationsregeln

Transformationsregel (template)

- Erzeuge aus Unterstruktur X im Ursprungsdokument Y im Ergebnisdokument!
- Beispiel:

```
<xsl:template match="order/item">
   <p><xsl:value-of select="."/></p>
</xsl:template>
```

=

```
<order>
  ...
  <item>Item</item>
  ...
</order>
```

** *Item* **

- Identifizierung von Unterstrukturen mit W3C-Standard XPath.

Ursprungs- und Ergebnisdokument

Freie Universität Berlin

```
<?xml version="1.0"?>
<order>
 <salesperson>John Doe</salesperson>
 <item>Production-Class Widget</item>
 <quantity>16</quantity>
 <date>...</date>
 <customer>Sally Finkelstein</customer>
</order>
```

```
<xsl:template match="order/item">
   <p><xsl:value-of select="."/></p>
</xsl:template>
```

Template

```
<p>Production-Class Widget</p>
```

Transformation

Ursprungsdokument → Ursprungsbaum (source document → source tree)

Ergebnisbaum → Ergebnisdokument (result tree → result document)

Weitere Programmierparadigmen

Freie Universität Berlin

XSLT-Transformationsregeln

- immer auf Ursprungsdokument(en) angewandt, niemals auf Zwischenergebnissen

- keine Seiteneffekte:
  - Template angewandt auf X liefert immer das gleiche Ergebnis
  - = Templates haben keine Zustände
  - ⇨ keine Variablen, die überschrieben werden können
  - ⇨ oft auch funktionales Programmierparadigma genannt

Grundstruktur von Stylesheets

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

- XML-Dokument
- Dokument-Wurzel:
  - stylesheet oder transform aus entsprechendem W3C-Namensraum
  - stylesheet und transform gleichbedeutend
  - obligatorisches Attribut: version

XPath

Freie Universität Berlin

- Standard zum Zugreifen beliebiger Teile eines XML-Dokumentes
- wird von XSLT benutzt
- Adressierungspfaden eines Dateisystems ähnlich:

  z.B. /order/item
- aber wesentlich mächtiger

- XPath 1.0 – W3C-Standard seit Nov. 1999
  - http://www.w3.org/TR/xpath

- XPath 2.0 – W3C-Standard seit Jan. 2007
  - http://www.w3.org/TR/xpath20/

Zugrundelegendes Dokumentenmodell

gleiches Modell wie in DOM

- XML-Dokument als Baum mit Elementen, Attributen und PCDATA als Knoten:
  - Element-Knoten
  - Attribut-Knoten
  - Text-Knoten

- virtuelle Dokument-Wurzel:

  durch " /" repräsentiert (links von "/" steht nichts)

⇨ Wurzel-Element immer Kind von " /":

z.B. /root

Zugriff auf Elemente und Attribute

Freie Universität Berlin

- Elemente werden einfach über ihren Namen identifiziert:

  z.B. **order** oder order/**item**

- Attribute werden mit "@name" identifiziert:

  z.B. **@id** oder order/**@id**

Absolute und relative Pfade

Freie Universität Berlin

absolute Pfade

- beginnen mit "/"
- z.B. /order/item

  ➔ lesen: Folge dem Pfad von der Dokument-Wurzel zu einem Kind-Element order und von dort aus zu einem Kind-Elementen item!

relative Pfade

- beginnen mit einem Element oder Attribut
- z.B. order/item

  ➔ lesen: item-Elemente, die Kind eines Elementes order sind
- Element order an beliebiger Stelle des XML-Dokumentes

Pfad-Ausdrücke

.	aktueller Knoten
..	Eltern-Knoten
*	beliebiges Kind-Element
@*	beliebiges Attribut
//	überspringt ≥ 0 Hierarchie-Ebenen nach unten
[]	spezifiziert ein Element
|	Auswahl (Vereinigung)

- Beispiel: *|@*

  „Kind-Element oder Attribut des aktuellen Knotens“

Kontext-Knoten

- XPath-Pfade werden in XSLT immer bzgl. eines bestimmten Kontext-Knotens ausgewertet:

  Element-, Attribut- oder Text-Knoten

- Beispiel:

```
<xsl:template match="p">
    <DIV>
      <xsl:value-of select="."/>
    </DIV>
</xsl:template>
```

- Was bedeutet hier aktueller Knoten "." ?
- "." = Kontext-Knoten
- Kontext-Knoten = Knoten, auf den das Template angewandt wird (hier ein p-Element)

Filter: Randbedingungen für Pfade

Freie Universität Berlin

- order/item[@item-id = 'E16-25A']

  item-Elemente, die Kind von order sind und Attribut item-id mit Wert 'E16-25A' haben

- können an beliebiger Stelle in einem Pfad vorkommen:

  order[@order-id = '4711']/item

XPath 1.0 Funktionen

Freie Universität Berlin

- Knoten-Set Funktionen
  - Beispiel:

    order/item[position() = 1]

    order/item[position()=last()]

- Boolesche Funktionen
  - boolean() – nimmt ein Objekt und liefert booleschen Wert zurück
  - not() – nimmt boolschen Ausdruck und liefert *true* wenn Argument ist *false*

  - Beispiel: order/item[not(position()=last())]

XPath 1.0 Funktionen

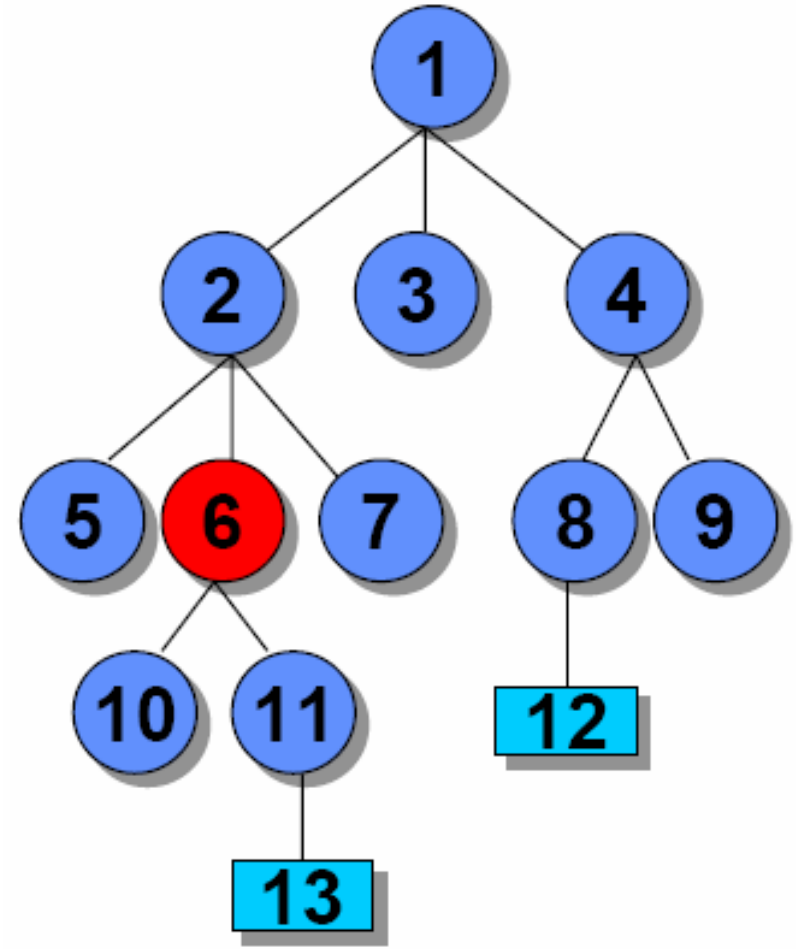
Freie Universität Berlin

- Numerische Funktionen
  - number() – versucht eine Zeichenkette als Zahl zu interpretieren und gibt die ermittelte Zahl zurück
  - sum() – Ermittelt die Gesamtsumme der Zahlenwerte des Ausgangsknotens

  - Beispiel: number(3xy) → 3

- String-Funktionen
  - string() - interpretiert ein übergebenes Argument als Zeichenkette und gibt die ermittelte Zeichenkette zurück
  - string-length() – liefer die Länge von String

XPath 1.0 Achsen



Quelle: http://swt.cs.tu-berlin.de/informatik2000/skripte/xml-datenbank.pdf

- self:: 6
- child:: 10, 11
- parent:: 2
- descendant:: 10, 11, 13
- descendant-or-self:: 6, 10, 11, 13
- ancestor:: 2, 1
- ancestor-or-self:: 6, 2, 1
- preceding-sibling:: 5
- preceding:: 5, 2, 1
- following-sibling:: 7
- following:: 10, 11, 13, 7, 3, 4, 8, 9, 12
- attribute:: Ø
- namespace:: Ø

Beispiele

Wähle das Wurzelelement AAA aus:

```
<AAA>
   <BBB/>
   <CCC/>
   <BBB/>
   <BBB/>
   <DDD>
      <BBB/>
   </DDD>
   <CCC/>
</AAA>
```

/AAA

Wähle alle CCC Elemente aus, die Kinder des Elements AAA sind:

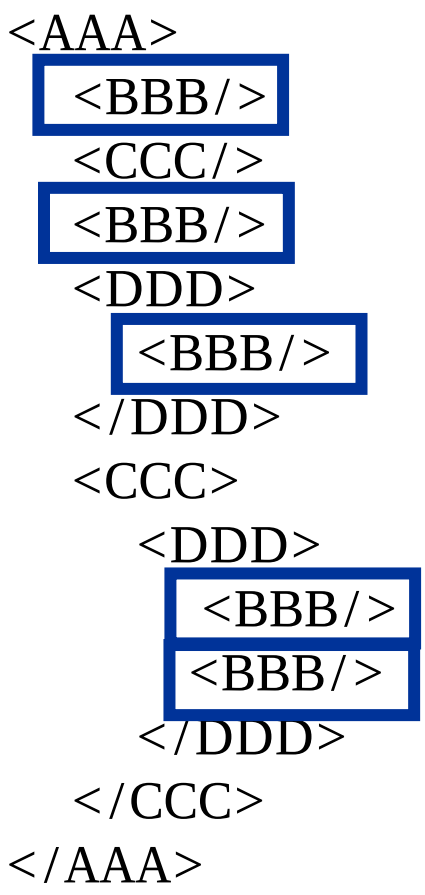
```
<AAA>
   <BBB/>
   <CCC/>
   <BBB/>
   <BBB/>
   <DDD>
      <BBB/>
   </DDD>
   <CCC/>
</AAA>
```

/AAA/CCC

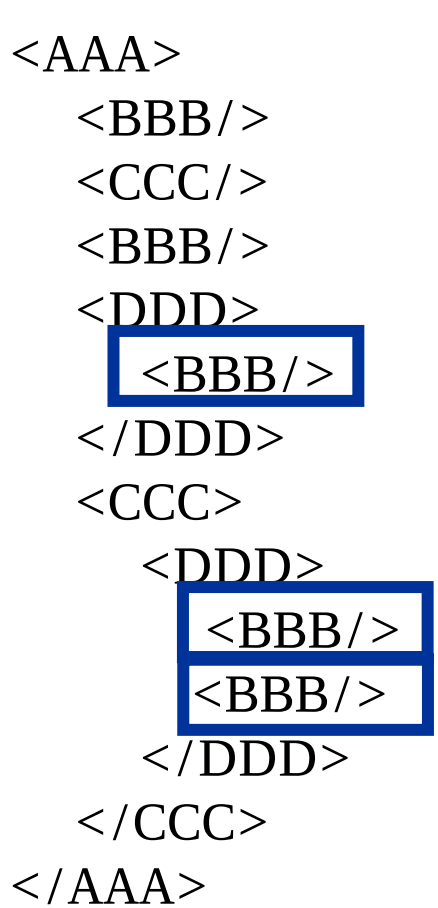
Beispiele

Freie Universität Berlin

//BBB



//DDD/BBB



Beispiele

Freie Universität Berlin

/*/*/*/BBB

```
<AAA>
  <XXX>
    <DDD>
      <BBB/>
      <FFF/>
    </DDD>
  </XXX>
  <CCC>
    <BBB>
      <BBB>
        <BBB/>
      </BBB>
    </BBB>
  </CCC>
</AAA>
```

//*

```
<AAA>
  <XXX>
    <DDD>
      <BBB/>
      <FFF/>
    </DDD>
  </XXX>
  <CCC>
    <BBB>
      <BBB>
        <BBB/>
      </BBB>
    </BBB>
  </CCC>
</AAA>
```

Beispiele

/AAA/BBB[last()]

```
<AAA>
   <BBB/>
   <BBB/>
   <BBB/>
   <BBB/>
</AAA>
```

//@id

```
<AAA>
   <BBB id = "b1" />
   <BBB id = "b2" />
   <BBB name = "bbb"/>
   <BBB/>
</AAA>
```

Beispiele

Freie Universität Berlin

//CCC | //BBB

```
<AAA>
  <BBB/>
  <CCC/>
  <DDD>
    <CCC/>
  </DDD>
  <EEE/>
</AAA>
```

//CCC/following-sibling::*

```
<AAA>
    <BBB>
      <CCC/>
      <DDD/>
    </BBB>
    <XXX>
      <EEE/>
      <CCC/>
      <FFF/>
      <FFF>
        <GGG/>
      </FFF>
    </XXX>
</AAA>
```

⇨ **http://www.futurelab.ch/xmlkurs/xpath.de.html**

XSLT: Templates

Inhalte erzeugen

- Template: „Suche im Ursprungsdokument Unterstruktur X und erzeuge hieraus im Ergebnisdokument Y!“

- zwei Möglichkeiten, Y zu erzeugen:
  1. neue Inhalte erzeugen
  2. Inhalte von X nach Y übertragen.
- beide Möglichkeiten beliebig miteinander kombinierbar

1. Neue Inhalte erzeugen

- Templates können alle XML-Inhalte erzeugen: PCDATA, Elemente und Attribute
- einfach normale XML-Syntax verwenden:

```
<xsl:template match="...">
   <p style="color:red">neuer Text</p>
</xsl:template>
```

- Beachte: Stylesheets müssen wohlgeformte XML-Dokumente sein, daher z.B. nicht erlaubt:

```
<xsl:template match="...">
   <br>neuer Text
</xsl:template>
```

1. Neue Inhalte erzeugen

- statt üblicher XML-Syntax

```
<xsl:template match="...">
   <p style="color:red">neuer Text</p>
</xsl:template>
```

- auch möglich:

```
<xsl:template match="...">
  <xsl:element name="p">
    <xsl:attribute
    name="style">color:red</xsl:attribute>
     <xsl:text>neuer Text</xsl:text>
    </xsl:element>
</xsl:template>
```

- nötig, wenn z.B. Name = Variable oder PCDATA = " "

2. Inhalte übertragen

Freie Universität Berlin

<xsl:copy-of select="."> Element

- Kopiert aktuellen Teilbaum
- aktueller Teilbaum: Baum, der vom aktuellen Knoten aufgespannt wird, einschließlich aller Attribute und PCDATA

<xsl:copy> Element

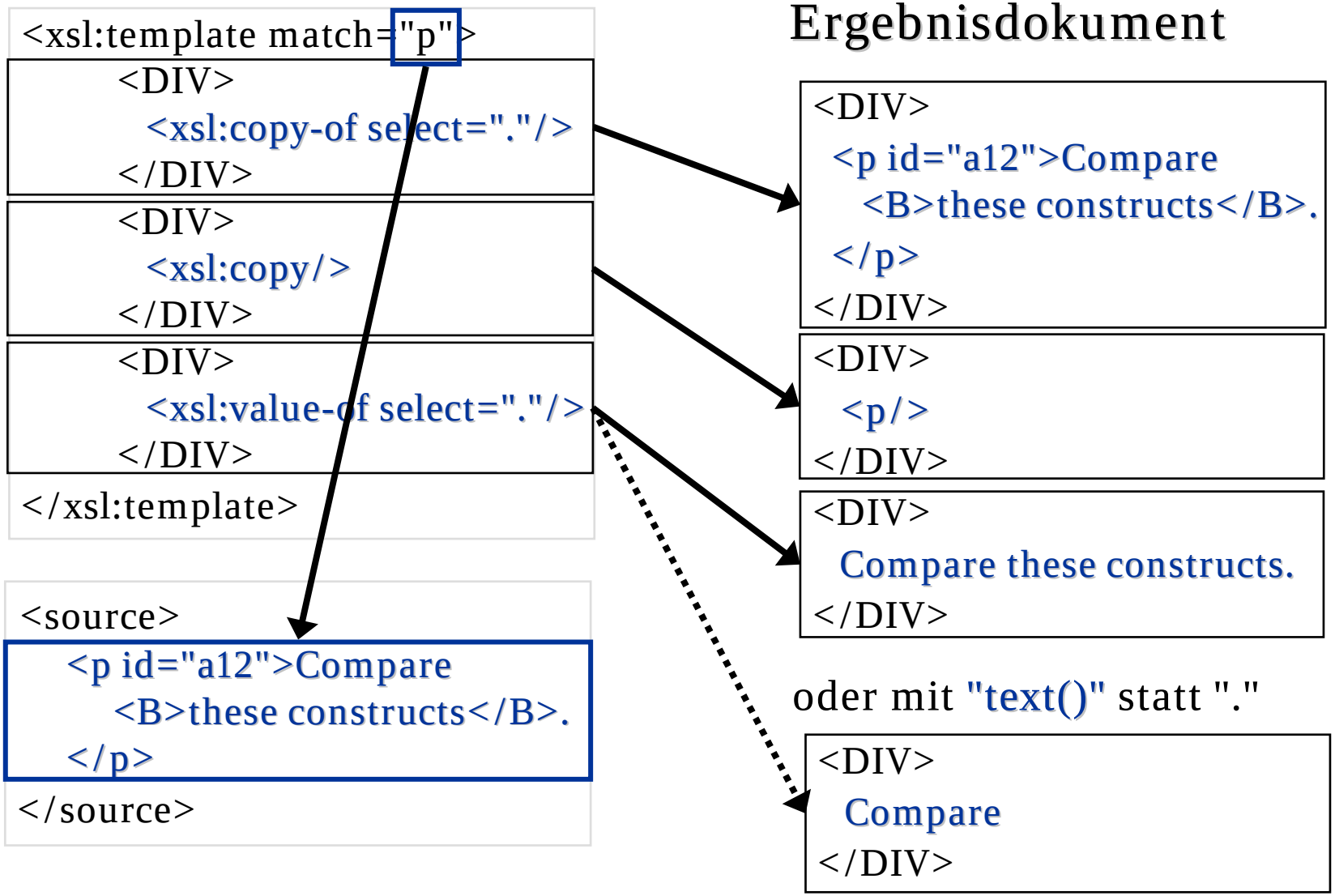
- Kopiert aktuellen Knoten ohne Kind-Elemente, Attribute und PCDATA

⇨ Kopiert nur Wurzel-Element des aktuellen Teilbaums

<xsl:value-of select="."> Element

- Extrahiert PCDATA, das im aktuellen Teilbaum vorkommt

Beispiel



Funktionsweise des XSLT-Prozessors

1. K := Dokument-Wurzel ("/") des Ursprungsdokumentes
2. Identifiziere alle Templates, die auf K anwendbar sind.
   a) Ist genau ein Template anwendbar, dann wende dieses an.

      Fertig.

   a) Sind mehre Templates anwendbar, dann wende das speziellste an:

      z.B. ist "/order" spezieller als "/*".

      Fertig.

   c) Ist kein Template anwendbar, dann wiederhole für alle Kinder K' von K Schritt 2 mit K := K'.

Beispiel

Freie Universität Berlin

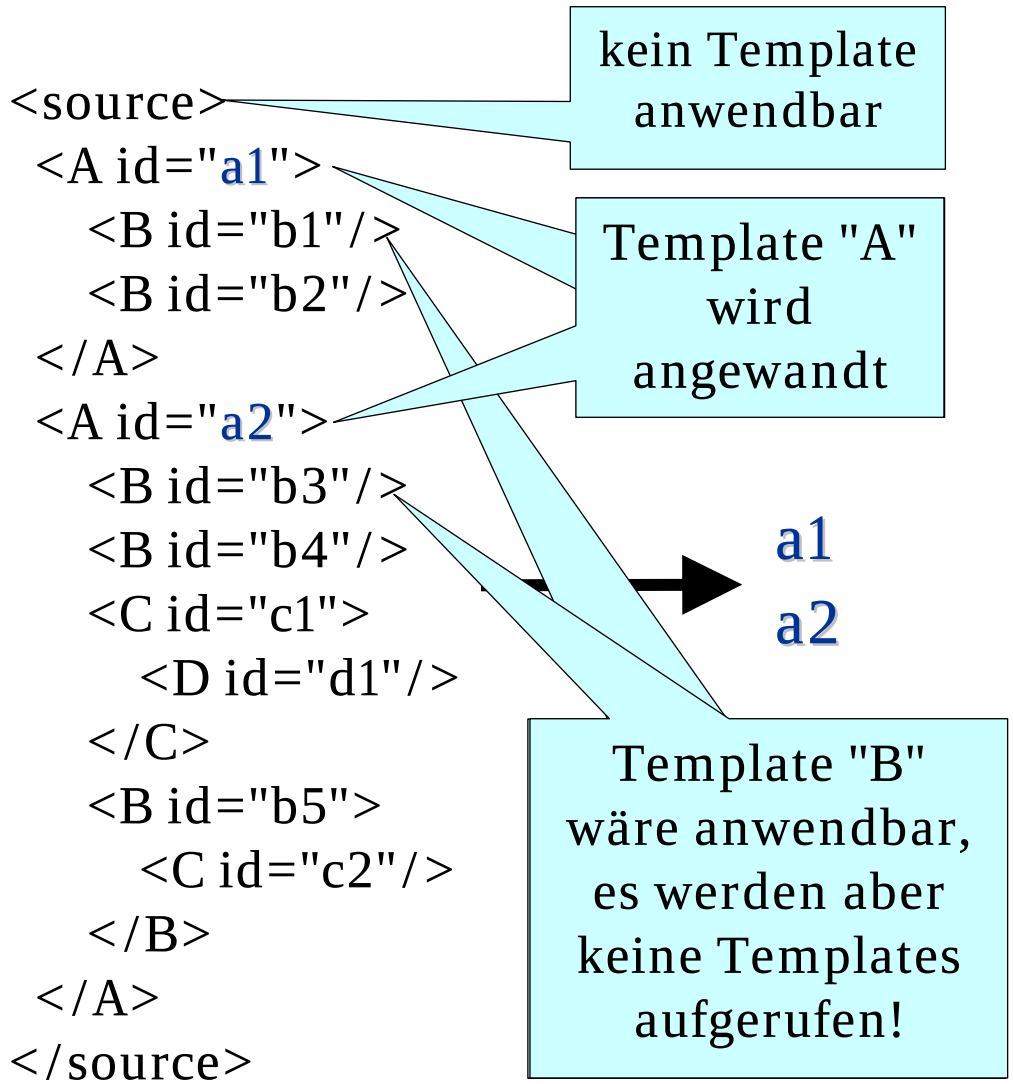
Stylesheet

```
<xsl:template match="A">
  <xsl:value-of select="@id"/>
</xsl:template>

<xsl:template match="B">
  <xsl:value-of select="@id"/>
</xsl:template>

<xsl:template match="C">
  <xsl:value-of select="@id"/>
</xsl:template>

<xsl:template match="D">
  <xsl:value-of select="@id"/>
</xsl:template>
```



Templates mit Rekursion

```
<xsl:template match="A">
  <xsl:value-of select="@id"/>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="B">
   <xsl:value-of select="@id"/>
   <xsl:apply-templates/>
 </xsl:template>

 <xsl:template match="C">
   <xsl:value-of select="@id"/>
   <xsl:apply-templates/>
 </xsl:template>

<xsl:template match="D">
  <xsl:value-of select="@id"/>
  <xsl:apply-templates/>
 </xsl:template>
```

```
<source>
 <A id="a1">
   <B id="b1"/>
   <B id="b2"/>
 </A>
 <A id="a2">
   <B id="b3"/>
   <B id="b4"/>
   <C id="c1">
     <D id="d1"/>
   </C>
   <B id="b5">
     <C id="c2"/>
   </B>
 </A>
</source>
```

```
a1
 b1
 b2
a2
 b3
 b4
 c1
  d1
 b5
  c2
```

Rekursiver Aufruf aller Templates

Freie Universität Berlin

<xsl:apply-templates/>

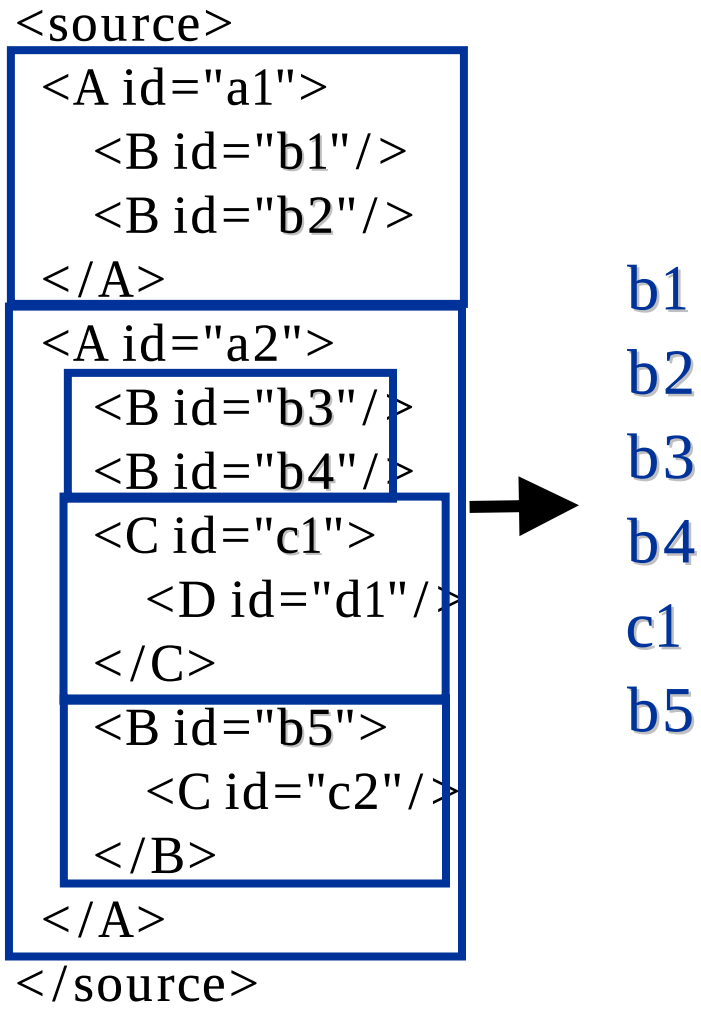
- versucht Templates auf Kinder des aktuellen Knotens anzuwenden
- Kind bedeutet hier: Kind-Element, Text-Knoten oder Attribut-Knoten
- Mit <xsl:apply-templates select = "…"/> auch rekursiver Aufruf an beliebiger Stelle möglich.
- Vorsicht: Terminierung nicht automatisch sichergestellt!
- Beispiel:

```
<xsl:template match="A">
  <xsl:value-of select="@id"/>
  <xsl:apply-templates select="/"/>
</xsl:template>
```

Iteration statt Rekursion

```
<xsl:template match="A">
  <xsl:for-each select="*">
    <xsl:value-of select="@id"/>
  </xsl:for-each>
</xsl:template>
```

- xsl:value-of wird auf alle select-Pfade der for-each-Schleife angewandt.
- Beachte: select-Pfad von xsl:for-each relativ zum Kontext-Knoten des Templates, hier also "A/*".



Zwei Vordefinierte Templates

Freie Universität Berlin

1. vordefiniertes Template

- realisiert rekursiven Aufruf des Prozessors, wenn kein Template anwendbar ist

2. vordefiniertes Template

- kopiert PCDATA und Attribut-Werte des aktuellen Knotens in das Ergebnisdokument

Leeres Stylesheet

- traversiert gesamtes Ursprungsdokument und extrahiert dabei PCDATA und Attribut-Werte

Überschreiben

- Vordefinierte Templates können durch speziellere Templates überschrieben werden.

1. vordefinierte Template

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

1. wird zuerst auf Dokument-Wurzel (" /") angewandt
2. versucht alle Templates anzuwenden
3. wird auf alle Kind-Elemente ("*") angewandt

- realisiert rekursiven Aufruf des XSLT-Prozessors
- wird von jedem speziellerem Template überschrieben: z.B. sind "/" und "item" spezieller als "*|/"
- spezielleres Template anwendbar ⇨ kein automatischer rekursiver Aufruf

2. vordefinierte Template

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

- Template wird auf PCDATA text() und Attribute @* angewandt
- text(): XPath-Funktion, selektiert PCDATA
- Template überträgt PCDATA bzw. Attribut-Wert in das Ergebnisdokument

Leeres Stylesheet

- Bei Stylesheet ohne Templates sind nur die beiden vordefinierten Templates aktiv:

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

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

- Gesamtes Ursprungsdokument wird traversiert, dabei wird PCDATA und Attribut-Werte extrahiert

Beispiel

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

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

```
<?xml version="1.0"?>
<name>
   <first>
     John
   </first>
   <middle>
     Fitzgerald Johansen
   </middle>
   <last>
     Doe
   </last>
</name>
```

match="/" ⇨ apply-templates

match="*" ⇨ apply-templates

match="*" ⇨ apply-templates

match="text()" ⇨ John

match="*" ⇨ apply-templates

match="text()" ⇨ Fitzgerald Johansen

match="*" ⇨ apply-templates

match="text()" ⇨ Doe

Identitäts-Stylesheet

- Stylesheet mit lediglich einem Template:

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

- wird auf jedes Element ("*") angewandt
- kopiert Wurzel des aktuellen Teilbaumes
- ruft rekursiv alle Templates auf

- überschreibt 1. vordefinierte Template <xsl:template match="*|/">, da spezieller
- Zusammen mit 2. vordefinierten Template <xsl:template match="text()|@*"> wird Ursprungsdokument kopiert.

Position des rekursiven Aufrufes?

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

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

```
<root>
 <a>aaa</a>
 <b>bbb</b>
 <c>ccc</c>
</root>
```

Ergebnis:

```
<root>
 <a>aaa</a>
 <b>bbb</b>
 <c>ccc</c>
</root>
```

```
<root/>
<a/>
aaa
<b/>
bbb
<c/>
ccc
```

Benannte Templates

- Templates können auch einen Namen haben:

```
<xsl:template match="/order/item" name="order-template">
   …
</xsl:template>
```

- Benannte Templates können gezielt mit

  `<xsl:call-template name="order-template"/>`

  aufgerufen werden.

Variablen

- Beispiel:

```
<xsl:variable name="X">
  <xsl:copy-of select="."/>
<xsl:variable>
```

- deklariert Variable X mit X := aktuellen Teilbaum
- Initiale Zuweisung kann nicht überschrieben werden!
- Wert von X: $X
- Beispiel:

```
<xsl:variable name="N">2</xsl:variable>
   ...
<xsl:value-of select="item[position()=$N]"/>
```

- Beachte: Variablen können nur global und nicht lokal in einem Template deklariert werden.

Parameter

- Templates können Parameter haben:

```
<xsl:template name="printRows">
  <xsl:param name="N"/>
    …
  <xsl:call-template name="printRows">
    <xsl:with-param name="N" select="$N + 1"/>
  </xsl:call-template>
</xsl:template>
```

Kontrollfluss: choose

Beispiel:

```
<xsl:template match="item">
 <part-number>
   <xsl:choose>
     <xsl:when test=". = 'Production-Class Widget'">
        E16-25A
     </xsl:when>
     <xsl:when test=". = 'Economy-Class Widget'">
        E16-25B
      </xsl:when>
     <xsl:otherwise>00</xsl:otherwise>
   </xsl:choose>
 </part-number>
</xsl:template>
```

Falls Inhalt von item = 'Production-Class Widget', dann erzeuge E16-25A

- Switch-Anweisung in Java ähnlich
- Abarbeitung von oben nach unten

Was bietet XSLT noch?

Freie Universität Berlin

Kontrollfluss

- <xsl:if test="*test*">*then*</xsl:if>

Sortieren

- <xsl:sort select="name/family"/>

Mehrere Ursprungsdokumente

- <xsl:apply-templates select="document('bib.xml')">

XPath-Funktionen

- <xsl:if test="not(position()=last())">...</xsl:if>

Und vieles mehr!

⇨ http://www.zvon.org/xxl/XSLTutorial/Output_ger/contents.html

Verarbeitung von Stylesheets

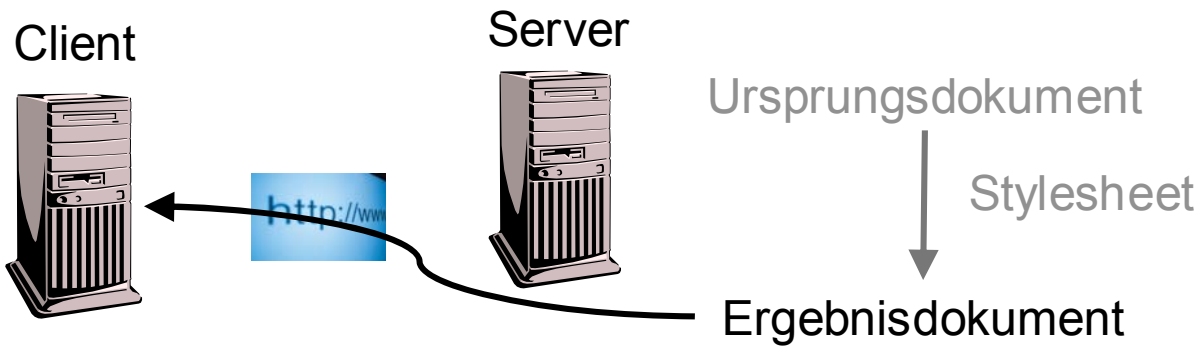
Freie Universität Berlin

Stylesheets können auf zwei Arten verarbeitet werden:

1. auf dem Server
2. im Client

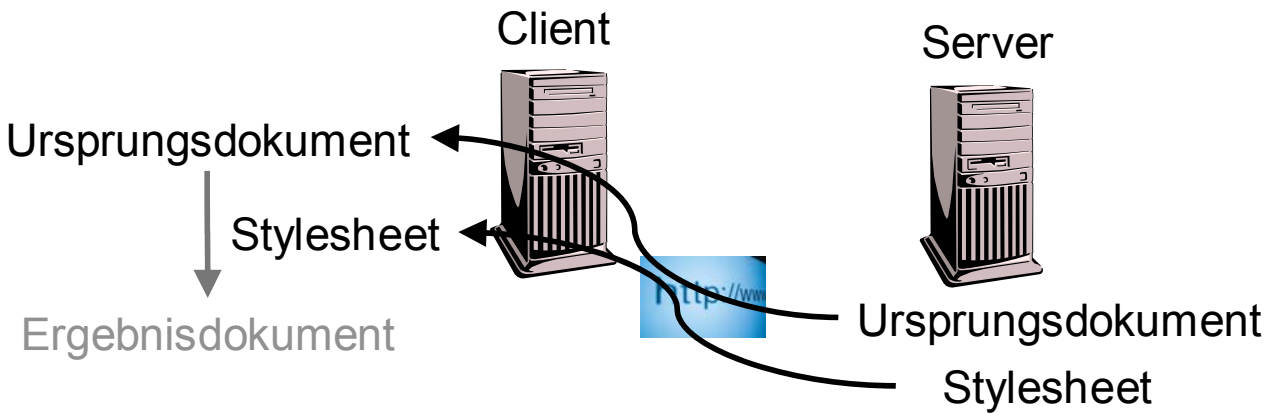
- Worin besteht der Unterschied?
- jeweiligen Vor- und Nachteile

1. Verarbeitung auf dem Server



- Server wendet passendes Stylesheet auf Ursprungs-dokument an.
- z.B. mit MSXML: msxsl *source stylesheet.xsl* –o *output*
- Client bekommt nur Ergebnisdokument

2. Verarbeitung im Client



- Client bekommt Ursprungsdokument & passendes Stylesheet.
- im Ursprungsdokument:

  <?xml-stylesheet type="text/xsl" href="stylesheet.xsl"?>
- Web-Browser wendet Stylesheet automatisch an und stellt Ergebnisdokument dar.

Wo Stylesheets verarbeiten?

Freie Universität Berlin

Verarbeitung im Client

+ Transformationen auf Clients verteilt: spart Server-Ressourcen
- Ursprungsdokument sichtbar

XSLT: stellt sicher, dass Transformation im Web-Client ausgeführt werden kann.

Verarbeitung auf dem Server

+ Ursprungsdokument verdeckt
- alle Transformationen auf zentralen Server

XSLT: nicht unbedingt nötig, da Transformation auf eigenem Server durchgeführt wird.

Mächtigkeit von XSLT

Freie Universität Berlin

- Variablen machen Stylesheets zu einem mächtigen Termersetzungssystem mit unbeschränkten Registern.
- www.unidex.com/turing definiert universelle Turingmaschine als XSLT-Stylesheet
  - Eingabe: Programm p (XML), Input i (XML)
  - Ausgabe: p(i)

⇨ Browser = vollwertigen Computer!

- Stylesheets tatsächlich berechnungsvollständig und damit vollwertige Programmiersprache (Kepser 2002)

➔ Terminierung von Stylesheets kann prinzipiell nicht garantiert werden.

Principle of Least Power

Freie Universität Berlin

When I designed HTML for the Web, I chose to avoid giving it more power than it absolutely needed - a "principle of least power," which I have stuck ever since. I could have used a language like Donald Knuth's "TeX," which though it looks like a markup language is in fact a programming language. It would allow you to express absolutely anything on the page, but would also have allowed Web pages that could crash, or loop forever (Tim Berner-Lees, 1999).

Verletzt XSLT dieses grundlegende Prinzip?

Vorteile von XSLT

Freie Universität Berlin

- plattformunabhäng
- relativ weit verbreitet
- Verarbeitung in Web-Browsern
- Standard-Transformationen (wie XML ➔ HTML) einfach zu realisieren.
- Nicht nur HTML, sondern beliebige andere Sprachen können erzeugt werden.
- extrem mächtig

Nachteile von XSLT

Freie Universität Berlin

- Entwickler müssen speziell für die Transformation von XML-Dokumenten neue Programmiersprache lernen.
- Anbindung von Datenbanken umständlich
- manche komplexe Transformationen nur umständlich zu realisieren.
- Terminierung kann nicht garantiert werden.

Fazit: XSLT nur für Standard-Transformationen verwenden!

XSL-FO

XSL Formatting Objects (XSL-FO)

Freie Universität Berlin

XSLT

- erlaubt Transformation von XML ➔ HTML
- ungeeignet für druckfähige Formatierungen (PDF, RTF)

XSL-FO

- erlaubt XML-Dokumente mit druckfähigen Layout zu versehen
- Transformation XML ➔ PDF oder RTF möglich
- basiert auf auf Cascading Style Sheets (CSS2)
- W3C-Standard von 2001

XSL = XSLT + XSL-FO

CSS vs. XSL-FO

Freie Universität Berlin

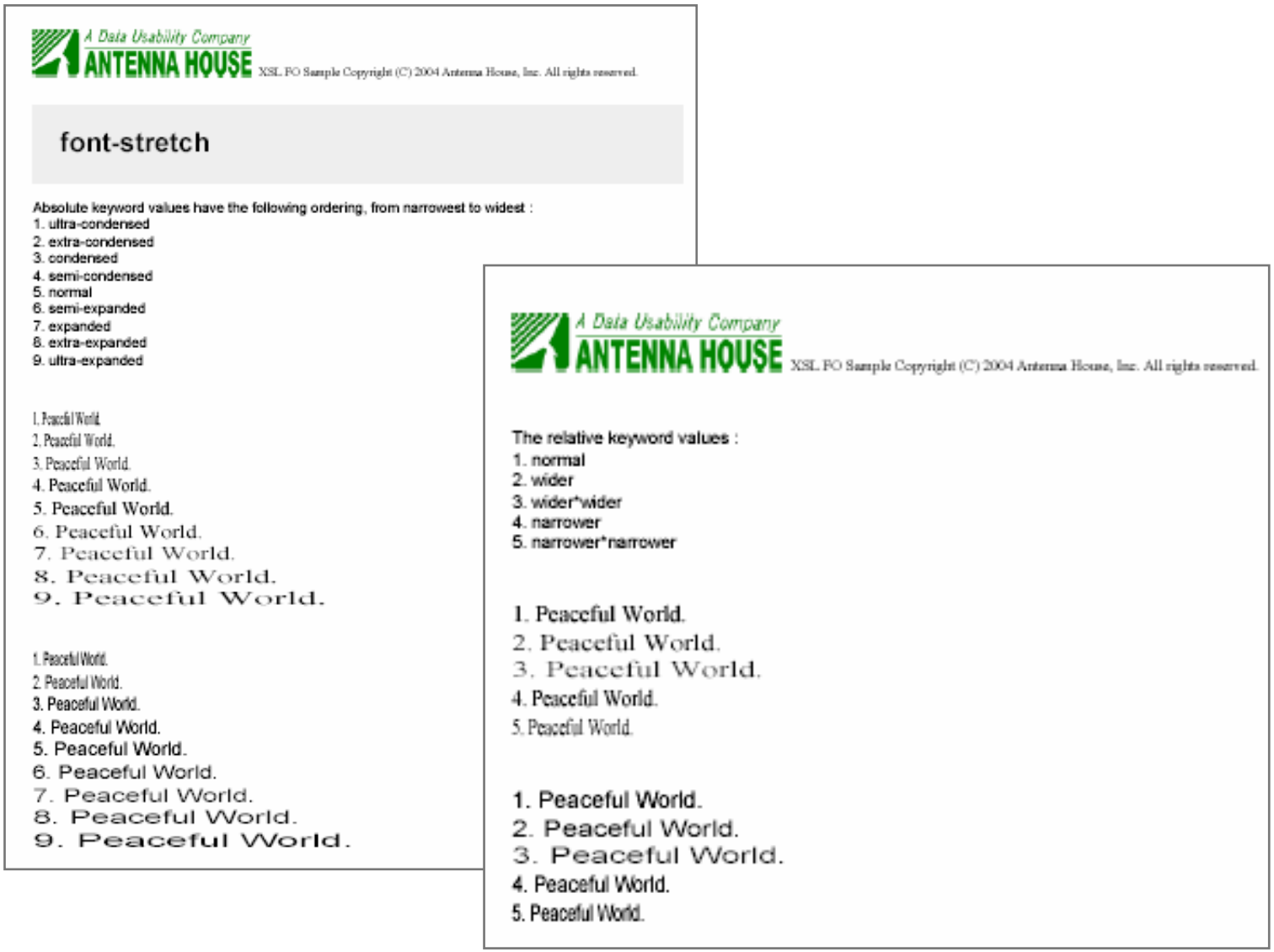
CSS	XSL-FO
Darstellung auf Bildschirm	Darstellung auf seitenorientiertem Ausgabemedium
Ausgabe durch Webbrowser	Ausgabe durch Drucker und andere Seitenausgabegeräte
Formatierungsinformation für vorhandenes Markup	Komplette Ersetzung von Markup durch - Formatierungsmarkup

XSL-FO Anwendungsgebiete

Freie Universität Berlin

- Massensatz, z.B.: bei der technischen Dokumentation
- gleichzeitige Ausgabe derselben Inhalte in unterschiedlichen Formaten:
  - verschiedene Medien
  - gleiches Medium aber verschiedene Bedürfnisse der Nutzer
- Individualisien bzw. Personalisieren von Dokumenten

Was leistet XSL-FO?



Wie sieht XSL-FO hierfür aus?

```
<?xml version="1.0" encoding="UTF-8" ?>
- <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format" xml:lang="en">
  - <fo:layout-master-set>
    - <fo:simple-page-master page-height="297mm" page-width="210mm" margin-top="10mm" margin-left="20mm" margin-right="20mm" margin-bottom="10mm" master-
      name="PageMaster">
        <fo:region-before extent="15mm" />
        <fo:region-body margin-top="20mm" margin-left="0mm" margin-right="0mm" margin-bottom="10mm" />
        <fo:region-after extent="15mm" />
      </fo:simple-page-master>
    </fo:layout-master-set>
  - <fo:page-sequence initial-page-number="1" master-reference="PageMaster">
    - <fo:static-content flow-name="xsl-region-before">
      - <fo:block font-size="9pt">
          <fo:external-graphic src="url('img/antenna-en.png')" content-height="12mm" />
          XSL FO Sample Copyright (C) 2004 Antenna House, Inc. All rights reserved.
        </fo:block>
      </fo:static-content>
    - <fo:flow flow-name="xsl-region-body">
        <fo:block text-indent="1em" font-family="Arial" font-size="20pt" font-weight="bold" background-color="#EEEEEE" line-height="20mm">font-stretch</fo:block>
        <fo:block start-indent="0em" font-family="Arial" font-size="10.5pt" linefeed-treatment="preserve">Absolute keyword values have the following ordering, from
          narrowest to widest : 1. ultra-condensed 2. extra-condensed 3. condensed 4. semi-condensed 5. normal 6. semi-expanded 7. expanded 8. extra-
          expanded 9. ultra-expanded</fo:block>
      - <fo:block font-family="Times New Roman" font-size="14pt" space-before="2em">
          <fo:block font-stretch="ultra-condensed">1. Peaceful World.</fo:block>
          <fo:block font-stretch="extra-condensed">2. Peaceful World.</fo:block>
          <fo:block font-stretch="condensed">3. Peaceful World.</fo:block>
          <fo:block font-stretch="semi-condensed">4. Peaceful World.</fo:block>
          <fo:block font-stretch="normal">5. Peaceful World.</fo:block>
          <fo:block font-stretch="semi-expanded">6. Peaceful World.</fo:block>
          <fo:block font-stretch="expanded">7. Peaceful World.</fo:block>
          <fo:block font-stretch="extra-expanded">8. Peaceful World.</fo:block>
          <fo:block font-stretch="ultra-expanded">9. Peaceful World.</fo:block>
        </fo:block>
      + <fo:block font-family="Arial" font-size="14pt" space-before="2em">
        <fo:block break-before="page" start-indent="0em" font-family="Arial" font-size="10.5pt" linefeed-treatment="preserve">The relative keyword values : 1. normal
          2. wider 3. wider*wider 4. narrower 5. narrower*narrower</fo:block>
      + <fo:block font-stretch="normal" font-family="Times New Roman" font-size="14pt" space-before="2em">
      + <fo:block font-family="Arial" font-size="14pt" space-before="2em">
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
```

Wie geht es weiter?

Freie Universität Berlin

heutige Vorlesung

- ☑ Warum XML-Dokumente transformieren?
- ☑ XSLT und XPath
- ☑ XSL-FO

Vorlesung nächste Woche

- XML & Datenbanken

Übung nächste Woche

- 4. Übung: XPath, XSLT