

---

# XSLT

## Heutige Vorlesung

---

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

# Warum XML-Dokumente transformieren?



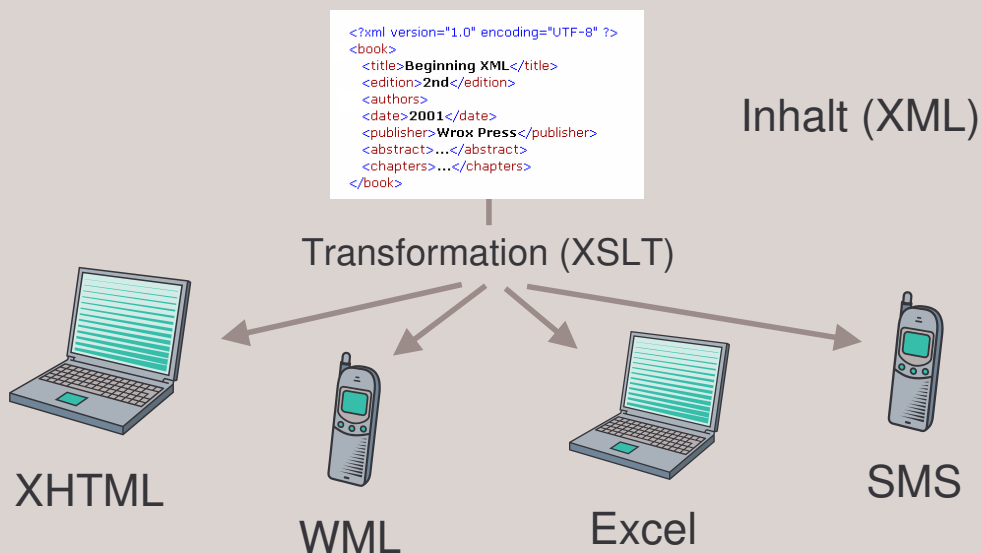
## Trennung Inhalt und Präsentation

- 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:** Trennung Inhalt von Präsentation
- Beachte: XHTML, WML, XSLT  $\subset$  XML

# XML-Inhalt → XML-Inhalt



Großhandel



Zulieferer



interner Kundenauftrag

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

externer Zulieferauftrag

- Datum
- Produktbezeichnung aus Zuliefererkatalog
- Anzahl
- Auftraggeber

*anpassen*

*übernehmen*

# XML-Inhalt → XML-Inhalt



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</customer>
</order>
```

Zulieferauftrag

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

andere Sicht (*view*)  
auf XML-Inhalt

---

# XSLT

## Was ist XSLT?

- spezielle 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

# Programmierparadigma von XSLT



## 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>
=
</order>
  ↓
<p>Item</p>
```

- XSLT benutzt zur Identifizierung von Unterstrukturen W3C-Standard XPath.

# Ursprungs- und Ergebnisdokument



```
<?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>
```

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

Ursprungsdokument  
(*source tree*)

Template

Ergebnisdokument

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

- XML-Dokument
- Dokument-Wurzel: entweder stylesheet oder transform aus entsprechendem W3C-Namensraum
- stylesheet und transform gleichbedeutend

# XPath

- Standard zum Zugreifen beliebiger Teile eines XML-Dokumentes
- W3C-Standard von 1999
- wird von XSLT benutzt
- Adressierungspfad eines Dateisystems ähnlich:  
z.B. /order/item
- aber wesentlich mächtiger

## Zugrundeliegendes 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 daher immer Kind von "/":  
z.B. /root

# Zugriff auf Elemente und Attribute



- 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



## absolute Pfade

- beginnen mit "/"
- z.B. **/order/item**  
→ lesen: Folge dem Pfad von der Dokument-Wurzel zu Kind-Element **order** und von dort aus zu 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




- . aktueller Knoten
- .. Eltern-Knoten
- \* beliebiges Kind-Element
- @\* beliebiges Attribut
- // überspringt  $\geq 0$  Hierarchie-Ebenen nach unten
- | Auswahl (Vereinigung)
- Beispiel: \*|@\*  
„Kind-Element oder Attribut des aktuellen Knotens“

- Pfade werden 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>
```

A black box highlights the text 'select=.'" in the XSL-FO code. A curved arrow points from this box to the text 'Was ist hier der aktuelle Knoten "." ?' in the list below.

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

- `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`
- können Vielzahl von Funktionen benutzen, z.B.:  
`order/item[position() = 1]`  
`order/item[position()=last()]`
- können Boolesche Ausdrücke enthalten:  
`order/item[not(position()=last())]`

# XSLT: Templates

- Template: „Suche im Ursprungsdokument bestimmte Unterstruktur X und **erzeuge hieraus im Ergebnisdokument Y!**“
- zwei verschiedene Möglichkeiten, Y zu erzeugen:
  1. Neue Inhalte erzeugen: PCDATA, Elemente oder Attribute.
  2. Inhalte von X nach Y übertragen.
- beide Möglichkeiten beliebig miteinander kombinierbar

## 1. Neue Inhalte erzeugen

- Templates können alle drei Kategorien von XML-Inhalten erzeugen: PCDATA, Elemente und Attribute
- Jeweils 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>
```

- daher hier Entity References `&lt;` und `&gt;` benutzen

```
<xsl:template match="order/item">
  <p style="color: red;">Item:
    &lt;br&gt;
  </p>
</xsl:template>
```

Template

Ursprungsdokument

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

Effekt des Templates

```
<p style="color: red;">
Item:
&lt;br&gt;
</p>
```

## 2. Inhalte übertragen: Drei Möglichkeiten

**<xsl:copy-of select="."/>**

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

**<xsl:copy/>**

- Kopiert aktuellen Knoten *ohne* Elemente, Attribute und PCDATA
- Kopiert also nur Wurzel-Element des aktuellen Teilbaums

**<xsl:value-of select="."/>**

- Extrahiert PCDATA aus aktuellen Teilbaum

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

```
<source>
  <p id="a12">Compare
    <B>these constructs</B>.
  </p>
</source>
```

## Effekt des Templates

```
<DIV>
  <p id="a12">Compare
    <B>these constructs</B>.
  </p>
</DIV>
```

```
<DIV>
  <p/>
</DIV>
```

```
<DIV>
  Compare these constructs.
</DIV>
```

oder mit "text()" statt "."

```
<DIV>
  Compare
</DIV>
```

# 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



## 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>
```

```
<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>
```

kein Template  
anwendbar

Template "A"  
wird  
angewandt

a1  
a2

Template "B"  
wäre anwendbar,  
es werden aber  
keine Templates  
aufgerufen!

# 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



```
<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 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 der Schleife relativ zum Kontext-Knoten des Templates, hier also "A/\*".

```
<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>
```

b1  
b2  
b3  
b4  
c1  
b5

## vordefiniertes Template 1

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

## vordefiniertes Template 2

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

## Leeres Stylesheet

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

## Überschreiben der vordefinierten Templates

- Vordefinierte Templates können durch speziellere Templates überschrieben werden.

# Erstes vordefinierte Template

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

1. wird zuerst auf Dokument-Wurzel (" /") angewandt
2. wendet rekursiv Templates auf Kinder an
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



## Zweites 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"?> match="/" ⇒ apply-templates
<name> match="*" ⇒ apply-templates
  <first> match="*" ⇒ apply-templates
    John match="text()" ⇒ John
  </first>
  <middle> match="*" ⇒ apply-templates
    Fitzgerald Johansen match="text()" ⇒ Fitzgerald Johansen
  </middle>
  <last> match="*" ⇒ apply-templates
    Doe match="text()" ⇒ Doe
  </last>
</name>
```

35

# Identitäts-Stylesheet

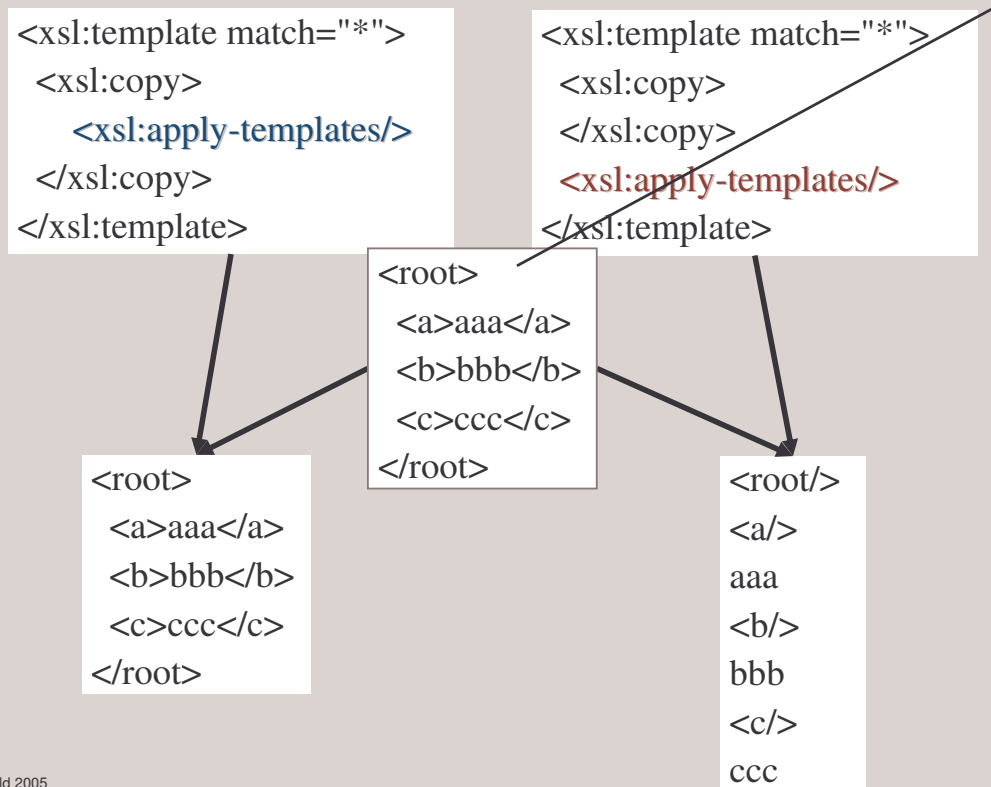


- Stylesheet mit lediglich einem Template:

```
<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
- Vordefiniertes Template `<xsl:template match="*/">` ist allgemeiner und wird deshalb überschrieben.
  - Zusammen mit zweiten vordefinierten Template `<xsl:template match="text()|@"*>` wird gesamtes Ursprungsdokument kopiert.

# Position des rekursiven Aufrufes?



# 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.

- Beispiel:

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

- Deklariert Variable X
- weist X den aktuellen Teilbaum (".") zu
- Beachte: Initiale Zuweisung kann *nicht* überschrieben werden!
- Variablen können als Parameter von Templates übergeben werden.

# 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

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

- Zugriff auf externe Datenbank wäre hier besser.
- In XSLT jedoch *nicht* möglich

- `xsl:import` erlaubt jedoch importieren externer Templates.
- mögliche Lösung: Datenbank → Templates → `xsl:import`

## Was bietet XSLT noch?

### Kontrollfluss: if

- `<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, vieles mehr!

# Verarbeitung von Stylesheets

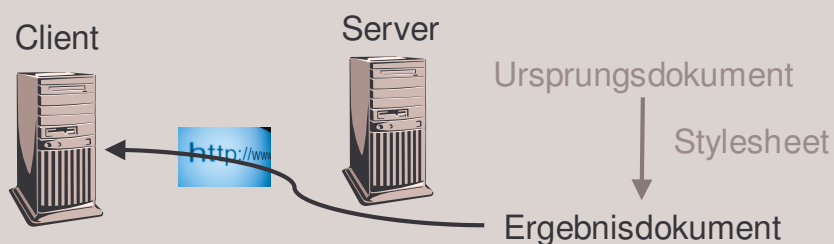


Stylesheets können auf zwei Arten verarbeitet werden:

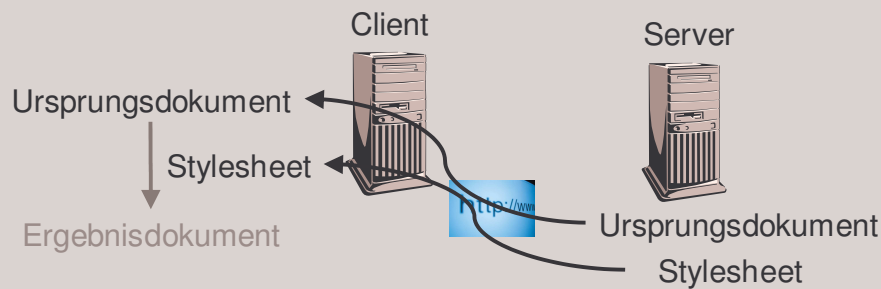
- auf dem Server
- im Client

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

# Verarbeitung auf dem Server



- Server wendet passendes Stylesheet auf Ursprungsdokument an.
- z.B. mit MSXML: `msxsl source stylesheet.xsl -o output`
- Client bekommt nur Ergebnisdokument



- Client bekommt Ursprungsdokument *und* 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?

### 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



- Variablen machen Stylesheets zu einem mächtigen Termersetzungssystem mit unbeschränkten Registern.
- [www.unidex.com/turing](http://www.unidex.com/turing) definiert universelle Turingmaschine als XSLT-Stylesheet
  - Eingabe: Programm  $p$  (XML), Input  $i$  (XML)
  - Ausgabe:  $p(i)$
- Damit wird der IE zum vollwertigen Computer!
- Stylesheets tatsächlich berechnungsvollständig und damit eine vollwertige Programmiersprache (Kepser 2002).

→ Terminierung von Stylesheets kann *prinzipiell* nicht garantiert werden.

# Principle of Least Power



*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



- + plattformunabhängig
- + 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



- Entwickler müssen *speziell* für die Transformation von XML-Dokumenten neue Programmiersprache lernen.
- Anbindung von Datenbanken umständlich
- komplexe Transformationen, wie Join von zwei Relationen nur umständlich zu realisieren.
- Terminierung kann nicht garantiert werden.

Fazit: XSLT nur für Standard-Transformationen verwenden!

---

# XSL-FO

## XSL Formatting Objects (XSL-FO)

### XSLT

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

### XSL-FO

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

**XSL = XSLT + XSL-FO**

# Was leistet XSL-FO?



XSL-FO Sample Copyright (C) 2004 Antenna House, Inc. All rights reserved.

## font-stretch

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

1. Peaceful World.
2. Peaceful World.
3. Peaceful World.
4. Peaceful World.
5. Peaceful World.
6. Peaceful World.
7. Peaceful World.
8. Peaceful World.
9. Peaceful World.

1. Peaceful World.
2. Peaceful World.
3. Peaceful World.
4. Peaceful World.
5. Peaceful World.
6. Peaceful World.
7. Peaceful World.
8. Peaceful World.
9. Peaceful World.



XSL-FO Sample Copyright (C) 2004 Antenna House, Inc. All rights reserved.

The relative keyword values :

1. normal
2. wider
3. wider\*wider
4. narrower
5. narrower\*narrower

1. Peaceful World.
2. Peaceful World.
3. Peaceful World.
4. Peaceful World.
5. Peaceful World.

1. Peaceful World.
2. Peaceful World.
3. Peaceful World.
4. Peaceful World.
5. Peaceful World.

# 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?

---



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

## heutiges Tutorium

- Beispiel XSLT: XML ⇔ XML
- Beispiel XSLT: XML ⇔ XHTML-Layout

## Vorlesung nächste Woche

- XML & Datenbanken