Transformation von XML-Dokumenten

© Klaus Schild 2003

Lernziele

- Warum XML-Dokumente transformieren?
- Wie funktioniert XSLT?
- Was ist XPath?
- Vor- und Nachteile von XSLT

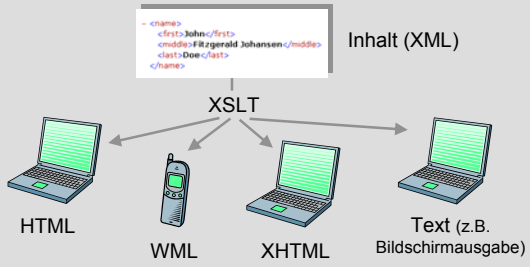
© Klaus Schild 2003

Was ist XSLT?

- XML trennt den Inhalt von der Präsentation (Layout).
- XSLT ermöglicht es, XML-Inhalte mit einem bestimmten Layout zu versehen.
- Mit XSLT können XML-Dokumente transformiert werden:
  - XML ➔ HTML
  - XML ➔ beliebiges Textformat
  - XML ➔ XML

© Klaus Schild 2003

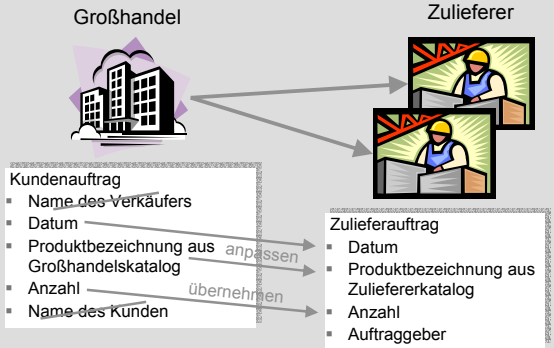
XML-Inhalt ➔ Layout



- *Multi-Delivery*: ein Inhalt, verschiedene Layouts
- Beachte: XHTML und WML sind XML-Standards.

© Klaus Schild 2003

XML-Inhalt ➔ XML-Inhalt



© Klaus Schild 2003

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</custome
</order>
```

XSLT

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>
```

© Klaus Schild 2003

Was ist XSLT?

- XSLT = Extensible Stylesheet Language Transformations
- W3C-Standard von 1999
- spezielle Programmiersprache zur Transformation von XML-Dokumenten
- XSLT-Programme (sog. *Stylesheets*) haben XML-Syntax
  - ➔ plattformunabhängig

© Klaus Schild 2003

Funktionsweise von XSLT

- Ein XSLT-Programm (*Stylesheet*) besteht aus einer Menge von Transformationsregeln (engl. *templates*) der Art:
- „Suche im Ursprungsdokument (engl. *source tree*) eine bestimmte Unterstruktur X und erzeuge hieraus im Ergebnisdokument Y!“
- Beispiel:

```
<xsl:template match="order/item">
   <p>Item encountered</p>
</xsl:template>
```

order/item

➔

Item encountered

© Klaus Schild 2003

Funktionsweise von XSLT

```
<xsl:template match="order/item">
   <p>Item encountered</p>
</xsl:template>
```

Template

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

Ursprungsdokument (*source tree*)

↓

```
<p>Item encountered</p>
```

Ergebnisdokument

© Klaus Schild 2003

Was ist ein Stylesheet?

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

Ein Stylesheet ist ein XSLT-Programm:

- Menge von Transformationsregeln (Templates)
- XML-Dokument
- Dokument-Wurzel: entweder stylesheet oder transform aus dem entsprechendem W3C-Namensraum.

© Klaus Schild 2003

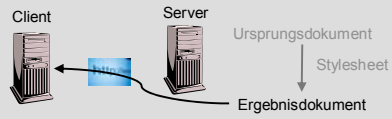
Verarbeitung von Stylesheets

Stylesheets können auf zwei Arten verarbeitet werden:

- auf dem Server
- im Client

© Klaus Schild 2003

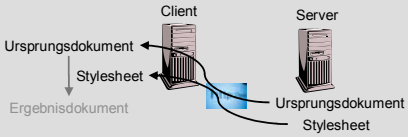
Transformation auf dem Server



- Server wendet Stylesheet auf das XML-Dokument an.
- MSXML: msxsl *source stylesheet* –o *output*
- unterschiedliche Stylesheets möglich
- Client bekommt nur das Ergebnisdokument.

© Klaus Schild 2003

Transformation im Client



- Server schickt dem Client das XML-Dokument zusammen mit dem Stylesheet.
- unterschiedliche Stylesheets möglich
- Im XML-Dokument ist das Stylesheet angegeben:
  <?xml-stylesheet type="text/xsl" href="*stylesheet*"?>
- Web-Browser wendet automatisch das Stylesheet an.

© Klaus Schild 2003

Verarbeitung von Stylesheets

Im Client:
+ Transformationen auf Clients verteilt (spart Server-Ressourcen)
- Ursprungsdokument sichtbar

XSLT stellt sicher, dass Transformation in allen Web-Clients ausgeführt werden kann.

Auf dem Server:
+ Ursprungsdokument verdeckt
- alle Transformationen auf zentralen Server

XSLT *nicht* nötig, Server kann jede andere Programmiersprache verwenden.

© Klaus Schild 2003

Templates: Inhalte suchen

- Grundprinzip: „*Suche im Ursprungsdokument eine bestimmte Unterstruktur X* und erzeuge hieraus im Ergebnisdokument Y!“
- Template sollte also beliebige Unterstrukturen eines XML-Dokumentes identifizieren können:

```
<xsl:template match="order/item">
  <p>Item encountered</p>
</xsl:template>
```

- XSLT benutzt hierfür den W3C-Standard XPath.

© Klaus Schild 2003

XPath

- W3C-Standard von 1999
- Standard zum Zugreifen beliebiger Teile eines XML-Dokumentes
- Adressierungspfaden eines Dateisystems ähnlich:
  z.B. /order/item

© Klaus Schild 2003

XPath: Virtuelle Dokument-Wurzel

- XML-Dokument wird als Parsebaum mit Elementen, Attributen und Text-Inhalten als Knoten interpretiert (wie in DOM).
- Eine virtuelle Dokument-Wurzel wird angenommen (ebenfalls wie in DOM).
- Dokument-Wurzel wird durch " /" repräsentiert (links von "/" steht nichts)
- Dokument-Typ ist daher immer Kind von "/":
  z.B. /order

© Klaus Schild 2003

XPath: Elemente und Attribute

- Elemente werden einfach mit ihrem Namen identifiziert:
  z.B. **order oder** order/**item**
- Attribute werden mit "@name" identifiziert:
  z.B. order/**@id**

© Klaus Schild 2003

XPath: Absolute und relative Pfade

absolute Pfade:

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

  „Folge den Pfad von der Dokument-Wurzel zu einem Kind-Element order und von dort aus zu einem Kind-Element item“

relative Pfade:

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

  „alle item-Elemente, die Kind des Elementes order sind“
- Element order an beliebiger Stelle des Dokumentes

© Klaus Schild 2003

XPath: Pfad-Ausdrücke

.	aktueller Knoten
..	Eltern-Knoten des aktuellen Knotens
*	beliebiges Kind-Element des aktuellen Knotens
@*	beliebiges Attribut des aktuellen Knotens
//	überspringt ≥ 0 Hierarchie-Ebenen
|	Auswahl (Disjunktion)

- Beispiel: ..|*|@*

  „Eltern-Element, Kind-Element oder Attribut des aktuellen Knotens“

© Klaus Schild 2003

XPath: Filter

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

  „alle item-Elemente, die Kind von order sind und ein Attribut item-id mit dem Wert 'E16-25A' haben“
- können an beliebiger Stelle in einem Pfad vorkommen:

  order**[@order-id = '4711']**/item
- können Vielzahl von Funktionen, wie z.B. text() benutzen:

  order/item[**text()** = 'E16-25A']
- text() liefert Text-Inhalt des aktuellen Knotens
- können beliebige Boolesche Ausdrücke enthalten:

  order/item[text() = 'E16-25A' **or** @item-id = 'E16-25A']

© Klaus Schild 2003

Templates: Inhalte erzeugen

- Grundprinzip: „Suche im Ursprungsdokument eine bestimmte Unterstruktur X und *erzeuge hieraus im Ergebnisdokument Y*!“
- Es gibt zwei verschiedene Möglichkeiten, Y zu erzeugen:
  1. Neue Inhalte (Text, Elemente oder Attribute) erzeugen.
  2. Inhalte von X nach Y übertragen.
- Beiden Möglichkeiten können beliebig miteinander kombiniert werden.

© Klaus Schild 2003

Neue Inhalte erzeugen

- Templates können alle drei Kategorien von XML-Inhalten erzeugen: Text, Elemente und Attribute.
- Jeweils einfach normale XML-Syntax verwenden:

```
<xsl:template match="order/item">
  <neu attr="value">neuer Text</neu>
</xsl:template>
```

© Klaus Schild 2003

Beispiel

```
<xsl:template match="order/item">
  <neu attr="value">neuer Text</neu>
</xsl:template>
```

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

Ergebnis

```
<neu attr="value">
  neuer Text
</neu>
```

© Klaus Schild 2003

Inhalte übertragen

drei Möglichkeiten, Inhalte zu übertragen:

- **<xsl:copy-of select="."/>** kopiert aktuellen Teilbaum vollständig.

  aktueller Teilbaum: Baum, der vom aktuellen Knoten aufgespannt wird, einschließlich aller Attribute
- <**xsl:copy**/> kopiert aktuellen Knoten *ohne* Attribute, Nachfolger oder Textinhalt (also nur Wurzel des aktuellen Teilbaums).
- **<xsl:value-of select="."/>** extrahiert alle Text-Inhalte aus aktuellen Teilbaum.

© Klaus Schild 2003

Beispiel

```
<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>
```

Ergebnis:

```
<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>
```

© Klaus Schild 2003

XSLT-Prozessor

- ausgehend von der Dokument-Wurzel wird versucht, auf einen Knoten K im Ursprungsdokument ein Template anzuwenden:
- Zuerst werden alle Templates identifiziert, die auf K anwendbar sind.
- Ist *genau ein* Template anwendbar, dann wird dieses angewandt.
- Sind *mehre* Templates anwendbar, dann wird das *speziellste* angewandt (z.B. ist "/order" spezieller als "/*").
- Ist *kein* Template anwendbar, so wird versucht, auf *jedes* Kind-Element von K ein Template anzuwenden.

© Klaus Schild 2003

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!

© Klaus Schild 2003

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

© Klaus Schild 2003

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 angewandt.
- Beachte: select-Pfad immer relativ zum match-Pfad, 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

© Klaus Schild 2003

Vordefinierte Templates

Es gibt zwei vordefinierte Templates:

- Eines für den rekursiven Aufruf des Prozessors, wenn kein Template auf aktuellen Knoten anwendbar ist.
- Ein zweites für die Extrahierung von Text-Inhalten und Attribut-Werten in das Ergebnisdokument.

→ Leeres Stylesheet traversiert das gesamte Ursprungsdokument und extrahiert alle Text-Inhalte und Attribut-Werte.

- Beide vordefinierten Templates können durch speziellere Templates überschrieben werden.

© Klaus Schild 2003

Erstes vordefinierte Template

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

1. wird zuerst auf Dokument-Wurzel " /" angewandt
2. ruft dann alle Templates (einschl. sich selbst) auf
3. wird anschließend auf alle Kind-Elemente "*" angewandt

- traversiert so das gesamte Ursprungsdokument und ruft jeweils alle Templates auf
- Beachte: wird z.B. von <xsl:template match="/"> und <xsl:template match="order"> überschrieben!

© Klaus Schild 2003

Zweites vordefinierte Template

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

- wird auf jeden Text-Knoten und jedes Attribut angewandt
- überträgt einfach Text-Inhalt bzw. Attribut-Wert in das Ergebnisdokument

© Klaus Schild 2003

Leeres Stylesheet

- Bei einem 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>
```

- Das gesamte Ursprungsdokument wird traversiert und dabei werden Text- und Attribut-Inhalte ausgegeben:

```
<name>
   <first>John</first>
   <middle>Fitzgerald Johansen</middle>
   <last>Doe</last>
</name>
```

→

```
John
Fitzgerald Johansen
Doe
```

© Klaus Schild 2003

Identitäts-Stylesheet

- Stylesheet mit lediglich einem Template:

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

- Dieses Template wird auf jedes Element angewandt.
- Vordefiniertes Template <xsl:template match="*|/"> ist allgemeiner und wird deshalb überschrieben.
- Zusammen mit dem zweiten vordefinierten Template <xsl:template match="text()|@*"> wird das gesamte Ursprungsdokument kopiert.

© Klaus Schild 2003

Benannte Templates

- Templates können auch einen Namen haben:

```
<xsl:template match="/order/item" name="order-template">
   <p>Item encountered</p>
</xsl:template>
```

- Benannte Templates können nicht nur mit xsl:apply-templates, sondern auch gezielt mit

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

  aufgerufen werden.

© Klaus Schild 2003

Kontrollfluss: if

```
<xsl:if test="order">
   Order encountered
</xsl:if>
```

- fragt hier die Existenz eines Kind-Elementes order ab
- test kann aber auch beliebige XPath-Ausdrücke (einschl. Boolescher Ausdrücke) enthalten:

```
<xsl:if test="number(employee/@FullSecurity)">
        <p><xsl:value-of select="employee/salary"/></p>
</xsl:if>
```

- fragt hier @FullSecurity > 0 ab

© Klaus Schild 2003

Kontrollfluss: choose

```
<xsl:choose>
  <xsl:when test="salary[number(.) > 2000]">
       A big number
  </xsl:when>
  <xsl:when test="salary[number(.) > 1000]">
       A medium number
  </xsl:when>
  <xsl:otherwise>A small number</xsl:otherwise>
</xsl:choose>
```

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

© Klaus Schild 2003

Beispiel

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</custome
</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>
```

© Klaus Schild 2003

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>
```

Ergebnis

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

© Klaus Schild 2003

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>
```

- Kein direkter Zugriff auf eine Datenbank möglich.
- xsl:import erlaubt jedoch importieren externer Templates.
- Lösung: Datenbank ➔ Templates ➔ xsl:import

© Klaus Schild 2003

Variablen

- Es können sowohl lokale, als auch globale Variablen definiert werden:

```
<xsl:variable name="X">
  <xsl:value-of select="/name/first">
<xsl:variable>
```

- Deklariert eine Variable X und weist ihr den Text-Inhalt des Knotens /name/first zu.
- Beachte: Initiale Zuweisung kann *nicht* überschrieben werden!
- Variablen können beliebige XML-Strukturen enthalten und als Parameter von Templates übergeben werden.

© Klaus Schild 2003

Mächtigkeit von XSLT

- Nicht nur HTML, sondern beliebige andere Sprachen können erzeugt werden.
- Variablen machen Stylesheets zu einem mächtigen Termersetzungssystem mit unbeschränkten Registern.
- www.unidex.com/turing definiert universelle Turingmaschine in XSLT.
- Damit wird der IE zum vollwertigen Computer!
- Stylesheets tatsächlich berechnungsvollständig und damit eine vollwertige Programmiersprache (Kepser 2002).

© Klaus Schild 2003

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?

© Klaus Schild 2003

Vorteile von XSLT

+ plattformunabhäng
+ relativ weit verbreitet
+ Standard-Transformationen (wie XML ➔ HTML) einfach (deklarativ) zu realisieren.
+ Nicht nur HTML, sondern beliebige andere Sprachen können erzeugt werden.
+ Stylesheets sind extrem flexibel und mächtig

© Klaus Schild 2003

Nachteile von XSLT

- Entwickler müssen *speziell* für die Transformation von XML-Dokumenten neue Programmiersprache lernen.
- Anbindung von Datenbanken umständlich
- Transformationen, die z.B. in SQL einfach (deklarativ) auszudrücken sind, nur umständlich (imperativ) zu realisieren.
- Stylesheets zu mächtig, um Transformationen (wie z.B. in SQL) automatisch zu optimieren.

Fazit: XSLT nur für Standard-Transformationen verwenden!

© Klaus Schild 2003