10.5 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)

10.5.1 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>
```

Inhalt eines CSS Stylesheet

- Liste von Paaren *Selektor* { *Properties* }
- Properties = List von Paaren Name der Property und ihr Wert: `color: green; border: solid red;`
- Selektor S besteht aus Elementspezifikationen E und Kombinationen
- Elementspezifikation E ist
  - `*`
  - ⟨*Elementname*⟩
  - E.⟨*Klassenname*⟩ (durch `class` Attribut)
  - E#⟨*Id*⟩ (durch `id` Attribut)
  - $E[A]$: Element mit Attribut A
  - $E[A = \langle String \rangle]$: mit Attribut A und Wert ⟨*String*⟩

Weitere Selektoren

- E: Elementspezifikation
- $S\ E$: E mit Vorfahren S
- $S{>}E$: E mit Vater S
- $S{+}E$: E mit Vorgängergeschwister S
- ...

Meist nur einfache Elementspezifikationen

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

10.5.2 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

  John Doe
  CEO, Widget Inc.
  john.doe@widget.inc
  (202) 555-1414

  - 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

10.5.3 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

Transformation durch template rules

- `<xsl:template>`
- pattern (`match` Attribut)
  - Vereinigung von Pfadausdrücken
  - Folge von Schritten ($\langle step \rangle$s) getrennt durch `/` und `//`
  - nur child und attribute Achsen (abgekürzt)
  - mit beliebigen Filtern
- replacement template (*sequence constructor*)
  - Erzeugt Folge von Element- und Textknoten
  - Konstruktion von Elementen und Attributen
  - Konstruktion von Text aus XML Datei
  - Aufruf von weiteren Regeln

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

10.5.4 XSLT Verarbeitungsmodell

- XSLT 2.0 verwendet XPath 2.0 für
  - Spezifikation von Mustern in template rules
  - Auswahl von Knoten zur Weiterverarbeitung
  - Bedingungen
  - Erzeugung von Textinhalten
- Auswerung relativ zu einem XPath-Kontext bestehend aus
  - Kontext Item (initial: Dokumentenwurzel)
  - Kontextposition und -größe (initial: beide 1)
  - Variablenbindungen (initial: globale Parameter)
  - Funktionsbibliothek (XPath Standardfunktionen, kein Namespacepräfix erforderlich)
  - Namespace-Deklarationen von Wurzelelement des Stylesheets

10.5.5 Transformationsregeln

The result tree is constructed by processing a sequence containing just the root node.

A sequence is processed by appending the sequences created by processing each of the members of the sequence in order.

Each node in the sequence is processed by finding all the template rules with patterns that match the node, and choosing the most specific amongst them;

the chosen rule's template is then instantiated with the node as the context node and with the sequence of source nodes as the rest of the focus.

A template typically contains instructions that select an additional sequence of source nodes for processing.

The process of matching, instantiation and selection is continued recursively until no new source nodes are selected for processing.

10.5.6 Syntax von Template-Regeln

```
<xsl:template
  match = "⟨pattern⟩"
  name = "⟨qname⟩"
  priority = "⟨number⟩"
  mode = "⟨qname⟩^+"
  as = "⟨sequence-type⟩">
  <!-- Content: (xsl:param*, template) -->
</xsl:template>
```

- `name` Aufruf als benannte Funktion
- `priority` überschreibt die Priorität des ⟨*pattern*⟩
- `mode` Gruppierung von Regeln, `#default`, `#all`
- `as` Typspezifikation

Beispiel

```
<xsl:template match="emph">
  <fo:inline-sequence font-weight="bold">
    <xsl:apply-templates/>
  </fo:inline-sequence>
</xsl:template>
```

transformiert

```
Nächste Woche betrachten wir die Generierung von
<emph>korrektem</emph> HTML.
```

in

```
Nächste Woche betrachten wir die Generierung von
<fo:inline-sequence font-weight="bold">korrektem
</fo:inline-sequence> HTML.
```

10.5.7 Sequenzkonstruktoren

- Rumpf von `<xsl:template>`
- Erzeugt Folge von Werten (entweder Atome oder Knoten)
- Konstruktoren für Elemente und Attribute (implizit oder explizit)
- Konstruktoren für Text
- Kopieren von Knoten
- Rekursiver Aufruf des Stylesheets
- Bedingte Konstruktion
- Expliziter Template-Aufruf

Elemente und Attribute

Implizite Konstruktion

```
<xsl:template match="/">
  <html>
    <head>
      <title>Hello World</title>
    </head>
    <body bgcolor="green">
      <b>Hello World</b>
    </body>
  </html>
</xsl:template>
```

Elemente und Attribute

Explizite Konstruktion

```
<xsl:template match="/">
  <xsl:element name="html">
    <xsl:element name="head">
      <xsl:element name="title">
        Hello World
      </xsl:element>
    </xsl:element>
    <xsl:element name="body">
      <xsl:attribute name="bgcolor" select="'green'"/>
      <xsl:element name="b">
        Hello World
      </xsl:element>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

- 'green' ist XPath-Ausdruck

Attribute Value Template

```
<xsl:variable name="image-dir">/images</xsl:variable>

<xsl:template match="photograph">
  <img src="{$image-dir}/{href}" width="{size/@width}"/>
</xsl:template>
```

- Globale Variable `<xsl:variable>` zur Abkürzung
- Attributwerte berechnet durch XPath-Ausdrücke in `{...}`
- Transformation von

```
<photograph>
  <href>headquarters.jpg</href>
  <size width="300"/>
</photograph>
```

nach

```
<img src="/images/headquarters.jpg" width="300"/>
```

Konstruktion von Text

- Literaler Text
- `<xsl:text>` konstanter Text (Leerzeichen im Rumpf bleiben erhalten)
- `<xsl:value-of select="..."/>` berechneter Text
- Literal + `value-of`

```
2 + 2 = <xsl:value-of select="2+2"/>
```

liefert

```
2 + 2 =4
```

- `text` + `value-of`

```
<xsl:text>2 + 2 = </xsl:text><xsl:value-of select="2+2"/>
```

liefert

```
2 + 2 = 4
```

Kopieren des Kontextknotens

```
<xsl:template match="ol|ul">
  <xsl:copy>
    <xsl:attribute name="style"
                   select="'list-style-type: square;'"/>
    <xsl:copy-of select="*"/>
  </xsl:copy>
</xsl:template>
```

- fügt jedem `<ol>` oder `<ul>` Knoten ein `style` Attribut hinzu
- `<xsl:copy>` erzeugt Kopie des Kontextknotens
  - ohne Kinder und Attribute
  - Inhalt spezifiziert neue Kinder und Attribute
- `<xsl:copy-of>` kopiert Knoten von Quelldokument

Rekursive Anwendung: `apply-templates`

```
<xsl:apply-templates
  select = "⟨node-set-expression⟩"
  mode = "⟨qname⟩">
  <!-- Content: (xsl:sort | xsl:with-param)* -->
</xsl:apply-templates>
```

- Wendet das ganze Stylesheet rekursiv auf ⟨*node-set-expression*⟩ an
- Voreinstellung `select="child::node()"`
- Optional
  - Wechselt nach Mode ⟨*qname*⟩
  - Parameterübergabe `<xsl:with-param>`
  - Sortieren

Beispiel

```
<xsl:template match="author-group">
  <fo:inline-sequence>
    <xsl:apply-templates select="author"/>
  </fo:inline-sequence>
</xsl:template>
```

```
<xsl:template match="product">
  <table>
    <xsl:apply-templates select="sales/domestic"/>
  </table>
  <table>
    <xsl:apply-templates select="sales/foreign"/>
  </table>
</xsl:template>
```

Beispiel mit Modes, I

```
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="students">
    <summary>
      <xsl:apply-templates mode="names" select="student"/>
      <xsl:apply-templates mode="grades" select="student"/>
    </summary>
  </xsl:template>

  <!-- siehe nächste Seite -->

  <xsl:template match="@grade">
    <grade>
      <xsl:value-of select="."/>
    </grade>
  </xsl:template>
</xsl:stylesheet>
```

Beispiel mit Modes, II

```
<xsl:template mode="names" match="student">
  <name>
    <xsl:attribute name="id" select="@id"/>
    <xsl:value-of select="name"/>
  </name>
</xsl:template>
<xsl:template mode="grades" match="student">
 <grades>
   <xsl:attribute name="id" select="@id"/>
   <xsl:apply-templates select=".//@grade"/>
  </grades>
</xsl:template>
```

Bedingte Inhalte

- Einfache Bedingung

```
<xsl:if test="⟨XPath-Ausdruck⟩">
  <!-- Sequenzkonstruktoren, falls Test wahr -->
</xsl:if>
```

- Mehrwegauswahl

```
<xsl:choose>
  <xsl:when test="⟨XPath-Ausdruck⟩">
    <!-- Sequenzkonstruktoren, falls Test wahr -->
  </xsl:when>
  <!-- weitere Tests mit xsl:when -->
  <xsl:otherwise>
    <!-- fall kein Test wahr -->
  </xsl:otherwise>
</xsl:choose>
```

10.5.8 Vordefinierte Regeln

- Suche rekursiv nach einer anwendbaren Regel

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

- Kopiere Textknoten und Attribute

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

- Ignoriere Verarbeitungsanweisungen und Kommentarknoten

```
<xsl:template match="processing-instruction()|comment()"/>
```

10.5.9 Funktionen, Aufrufe und Parameter

- Lokale Variable (gültig für den Rest eines Sequenzkonstruktors) gesetzt durch `<xsl:variable>` mit
  - `select`-Attribut oder
  - Sequenzkonstruktor im Rumpf
- Parameter für Templateaufrufe
  - Formaler Parameter (innerhalb von `<xsl:template>`) `<xsl:param>` (Voreinstellung wie bei `<xsl:variable>`)
  - Aktueller Parameter (innerhalb von `<xsl:apply-templates` oder `<xsl:call-template>`) `<xsl:with-param>`
- XSLT 1.0 kann keine Funktionen auf Funktionsergebnisse anwenden

Beispiel: Fakultätsfunktion

```
<xsl:variable name="input" select="100"/>

<xsl:template match="/">
  <html>
    <body bgcolor="#FFFFFF">
      <xsl:value-of select="$input"/>!
      <xsl:text> = </xsl:text>
      <xsl:call-template name="factorial">
        <xsl:with-param name="n" select="$input"/>
      </xsl:call-template>
    </body>
  </html>
</xsl:template>
```

Fakultätsfunktion, II

```
<xsl:template name="factorial">
  <xsl:param name="n"/>
  <xsl:param name="r" select="1"/>
  <!-- factorial n=<xsl:value-of select="$n"/>
                  r=<xsl:value-of select="$r"/> -->
  <xsl:choose>
    <xsl:when test="$n &lt;= 0">
      <xsl:value-of select="$r"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="factorial">
        <xsl:with-param name="n" select="$n -1"/>
        <xsl:with-param name="r" select="$n * $r"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

10.5.10 Typisches Muster: Einfaches Traversieren

```
<xsl:template match="@*|*">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>

<xsl:template match="ol|ul">
  <xsl:copy>
    <xsl:attribute name="style" select="'list-style-type: square;'"/>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>
```

10.5.11 Typisches Muster: Einfaches Traversieren, II

```
<xsl:template match="i">
  <em>
    <xsl:apply-templates select="@*|node()"/>
  </em>
</xsl:template>

<xsl:template match="b">
  <strong>
    <xsl:apply-templates select="@*|node()"/>
  </strong>
</xsl:template>

<xsl:template match="text()">
  <!-- this is an XPath 2.0 function -->
  <xsl:value-of select="upper-case(.)"/>
</xsl:template>
```

10.5.12 Formatierungsobjekte: XSL-FO

- XML-Sprache zur Spezifikation von gedrucktem Text
- Physikalisches Layout
- Bisher im Web nicht durchgesetzt
- Verwendet für Druckmedien

Beispiele für FO-Elemente

fo:root The fo:root node is the top node of an XSL result tree. This tree is composed of formatting objects.

fo:block The fo:block formatting object is commonly used for formatting paragraphs, titles, headlines, figure and table captions, etc.

fo:inline-graphic The fo:inline-graphic flow object is used for an inline graphic.

fo:list-block The fo:list-block flow object is used to format a list item or a list.

fo:list-item The fo:list-item formatting object contains the label and the body of an item in a list.

fo:list-item-body The fo:list-item-body formatting object contains the content of the body of a list item.

fo:list-item-label The fo:list-item-label formatting object contains the content of the label of a list item; typically used to either enumerate, identify or adorn the list-item's body.

fo:page-number The fo:page-number formatting object is used to obtain the current page-number.

fo:simple-link The fo:simple-link is used for representing the start resource of a simple link.

fo:table The fo:table flow object is used for formatting the tabular material of a table.

fo:table-body The fo:table-body formatting object is used to contain the content of the table body.

fo:table-caption The fo:table-caption formatting object is used to contain block-level formatting objects containing the caption for the table.

fo:table-cell The fo:table-cell formatting object is used to group content to be placed in a table-cell.

fo:table-column The fo:table-column formatting object specifies characteristics applicable to table cells that have the same column and span.

fo:table-footer The fo:table-footer formatting object is used to contain the content of the table footer.

fo:table-header The fo:table-header formatting object is used to contain the content of the table header.

fo:table-row The fo:table-row formatting object is used to contain the content of a table row.

10.5.13 Zusammenfassung

- XSL =
  - XSLT (XML Transformation)
  - XSL-FO (Druckformatierung)
- Inspiriert von CSS
- XSLT ist deklarative Sprache
  - Patternmatching
  - Sequenzkonstruktion
- XPath wesentlicher Bestandteil
  Knotenselektion, Patternmatching, Bedingungen, Strings