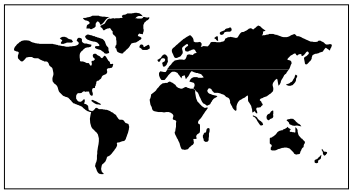
WWW – Grundlagen und Technologie

XML Path Language (XPath) und XSL Transformations (XSLT)

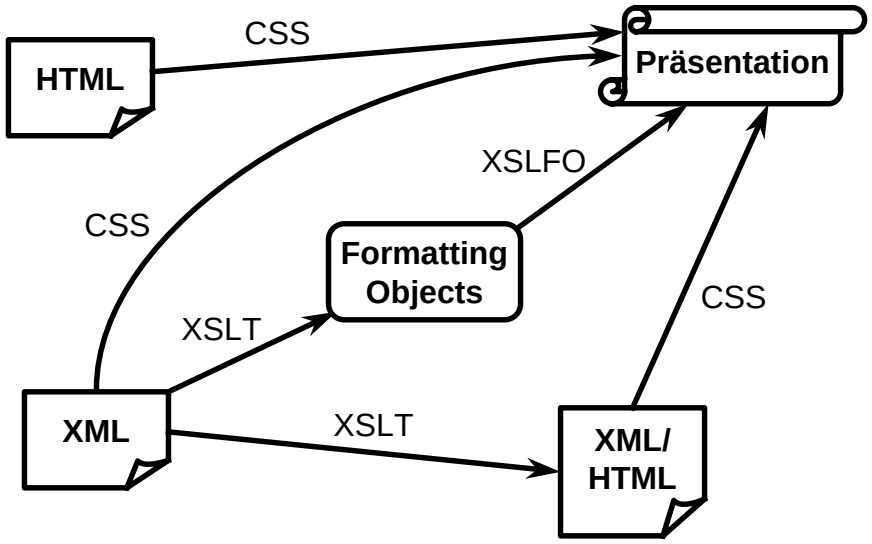


Erik Wilde
TIK – ETH Zürich
Sommersemester 2001

Übersicht

- *Extensible Stylesheet Language (XSL)*
- *XSL Transformations (XSLT)*
- *XML Path Language (XPath)*
- Zusammenfassung

Verwendung von Style Sheets



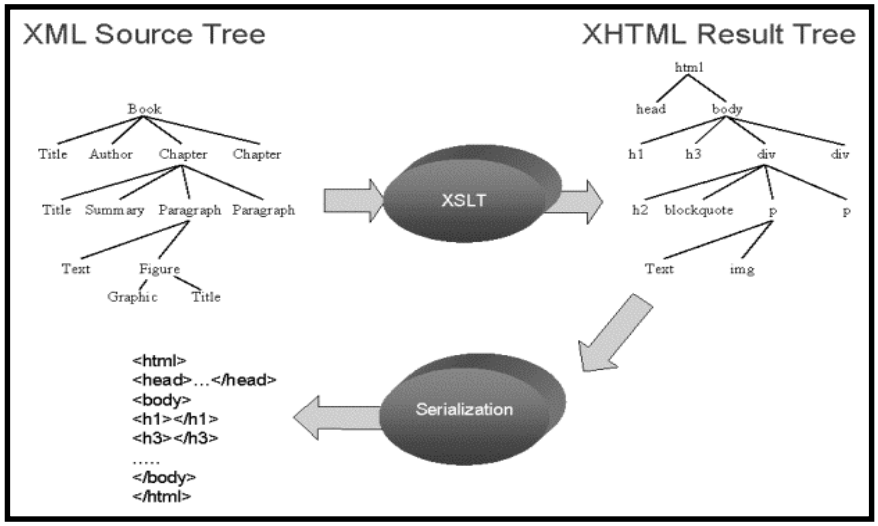
XSL Transformations (XSLT)

- Programmiersprache in XML
  - basierend auf Scheme
  - ...das seinerseits auf Lisp basiert
  - bestens geeignet für XML-Syntax (viele Klammern)
- keine *general purpose* Programmiersprache
  - Programmierstil vornehmlich deklarativ
  - prozedural möglich, aber nicht die Idee
  - mehr Logik im Laufzeitsystem als üblich
- generell Sprache für Transformationen
  - XML-spezifisches *Pattern Matching* (XPath)

Formatierung des XSLT Result Tree

- Orientierung am ISO-Standard
  - SGML als Sprache für strukturierte Dokumente
  - DSSSL als Sprache für Transformationen
  - SPDL als Sprache für die Seitenbeschreibung
- XSLFO wird der Standard zur Formatierung
  - momentan (02/00) noch im Draft Status
  - fehlende Funktionalität (Interface-Elemente)
- HTML als bekannte Seitenbeschreibungssprache
  - IE5 unterstützt nur die Formatierung in HTML
  - für viele Anwendungsfälle vorerst ausreichend
  - spätere Konvertierung möglich (XML bleibt gleich)

XML-X(HT)ML Transformation



Beispiel für Literal Result Elements

```
<xsl:template match="referent">
  <h3><a href="mailto:{@email}">
  <xsl:value-of select="vorname" />
  <xsl:value-of select="name" /></a>,
  <xsl:choose>
    <xsl:when test="organisation/@homepage">
      <a href="{organisation/@homepage}">
        <xsl:value-of select="organisation" /></a>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="organisation" />
    </xsl:otherwise>
  </xsl:choose>
  </h3>
</xsl:template>
```


XPath

- *Expression*, die ein *Objekt* ergibt
  - was ist das Ergebnis der Auswertung
  - in welchem Kontext findet die Auswertung statt
- XPath wird mehrfach verwendet
  - in *XSLT* für die Selektion für Style Sheets
  - in *XPointer* für die Selektion für URI Referenzen
  - wahrscheinlich im Standard für *XML Query*
- XPath stellt somit Grundlagenwissen dar
  - allerdings: anwendungsspezifische Erweiterungen
  - XSLT und XPointer verwenden XPath-Erweiterungen

Beispiel für XPaths

```
<xsl:template match="referent">
  <h3><a href="mailto:{@email}">
  <xsl:value-of select="vorname" />
  <xsl:value-of select="name" /></a>,
  <xsl:choose>
    <xsl:when test="organisation/@homepage">
      <a href="{organisation/@homepage}">
        <xsl:value-of select="organisation" /></a>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="organisation" />
    </xsl:otherwise>
  </xsl:choose>
  </h3>
</xsl:template>
```


Resultat der XPath Anwendung

```
<h3>
<a href="mailto:xml@dret.net">Erik
  Wilde</a>,
<a href="http://www.tik.ee.ethz.ch/">ETH
  Zürich</a>
</h3>
```

XPath Location Path

- Folge von *location steps*, getrennt mit `/`
  - erste Komponente: *Axis specifier* (gefolgt von `::`)
  - zweite Komponente: *Node Test*
  - dritte Komponent: 0-n Prädikate (in `[]`)
- es existieren eine Reihe von Abkürzungen
  - `child::` ist die *Default Axis*
  - `attribute::` kann abgekürzt werden als `@`
  - `//` ist kurz für `/descendant-or-self::node()/`
  - `*` selektiert alle Kinder-Elemente des Kontext-Node
  - `.` ist kurz für `self::node()`
  - `..` ist kurz für `parent::node()`

Beispiele für XPath Expressions

- `child::para` selects the para element children of the context node
- `descendant::para` selects the para element descendants of the context node
- `self::para` selects the context node if it is a para element, and otherwise selects nothing
- `child::chapter/descendant::para` selects the para element descendants of the chapter element children of the context node
- `child::para[position()=last()]` selects the last para child of the context node
- `/descendant::figure[position()=42]` selects the forty-second figure element in the document
- `/child::doc/child::chapter[position()=5]/child::section[position()=2]` selects the second section of the fifth chapter of the doc document element

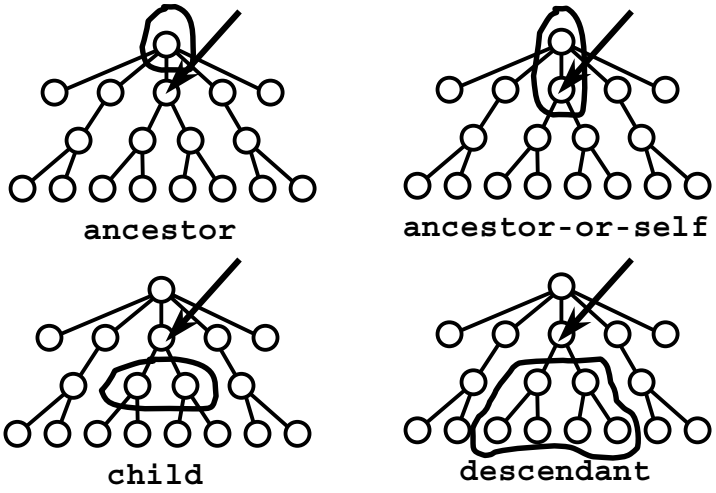
Beispiele für XPath Expressions (II)

- `para` selects the para element children of the context node
- `*` selects all element children of the context node
- `@name` selects the name attribute of the context node
- `para[last()]` selects the last para child of the context node
- `/doc/chapter[5]/section[2]` selects the second section of the fifth chapter of the doc
- `//para` selects all the para descendants of the document root and thus selects all para elements in the same document as the context node
- `para[@type="warning"][5]` selects the fifth para child of the context node that has a type attribute with value warning
- `para[5][@type="warning"]` selects the fifth para child of the context node if that child has a type attribute with value warning

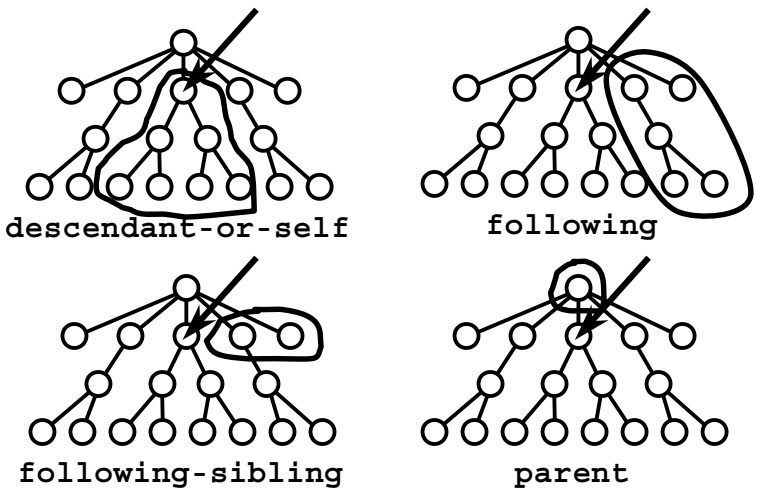
Abarbeitung eines Location Path

1. Feststellen des initialen Kontext
2. Konstruieren eines Node-Set
3. Anwenden eines Location Step, als Ergebnis ergibt sich ein neuer Kontext
4. Wiederholen dieser Schritte, bis alle Location Steps abgearbeitet sind
5. Das Endresultat ist das Node-Set für den gesamten Location Path

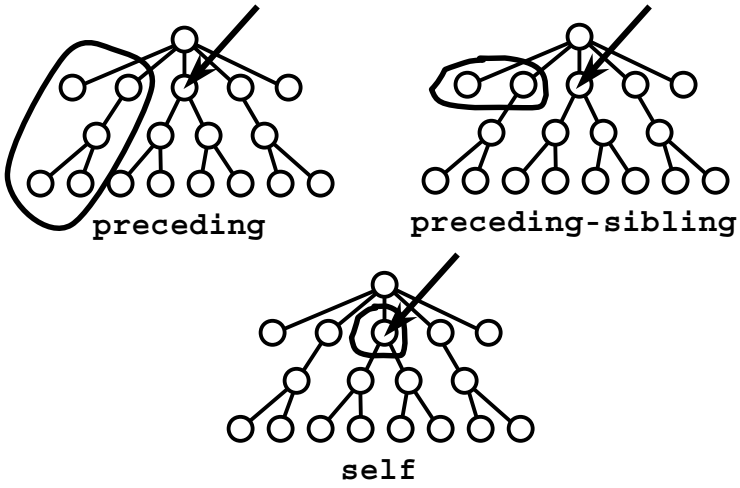
XPath Axes (I)



XPath Axes (II)



XPath Axes (III)



XPath Axes (IV)

- die ersten elf Axes "sehen" primär Elemente
  - Unterteilung in *forward* und *reverse*
  - Navigation im Dokumentenbaum
- weitere Axis für alle Attribute eines Elements
  - Zugriff auf einzelne Attribute
  - *Attribute Nodes* hängen an *Element Nodes*
- weitere Axis für alle Namespaces
  - ermöglicht Selektion von bestimmten Namespaces
  - *Namespace Nodes* hängen an *Element Nodes*
  - z.B. Trennung von Standard und interner DTD

XPath Node Tests

- ein Namenstest
  - testet einen Node auf einen bestimmten Namen
  - es wird einfach der Name angegeben
  - einfachster Test: **`/kurs/referent/name`**
  - Verwendung von Namespaces ist möglich
  - **`*`** kann verwendet werden als *Wildcard Character*
- eine Test auf den Nodetype
  - markiert mit nachfolgenden Klammern
  - Test auf **`text()`**, **`comment()`** oder **`node()`**
- spezielle *Processing Instruction*
  - im Dokument markiert mit **`<? ... ?>`**
  - anwendbar auf spezifische Processing Instruction
  - **`processing-instruction("myapplication")`**

XPath Prädikate

- weitere Einschränkung der Resultate
  - Selektion einer Axis (wählt die Kandidaten)
  - Spezifikation eines Node Test (Auswahl)
  - Prädikate filtern Node-Sets
- jeder Node im Node-Set wird getestet
  - je nach Resultat im *Result Set* oder nicht
- angegeben in **`[ ... ]`** nach dem Node Test
  - Inhalt ist eine *Expression*
  - Ergebnis wird nach Boolean konvertiert
- verwenden meistens Funktionen
  - *Core Functions* und Standard-spezifische Funktionen

XPath ist keine Query-Sprache

- Query-Sprachen kennen zwei Aspekte:
  - Adressierung von gesuchten Informationen
  - Manipulation der gesuchten Informationen
  - XPath behandelt nur den ersten Aspekt!
- XQL (verwendet von Software AG's Tamino)
  - war die Grundlage für XPath
  - geht über XPath hinaus
- XML Query als neue Query-Sprache für XML
  - sollte auf XPath aufbauen
- XPath als Grundlage für viele Standards
  - XSLT, XPointer, XML Query (XQL)

Aussehen eines XSLT Templates

Most templates have the following form:

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

- The whole `<xsl:template>` element is a *template*
- The *match pattern* determines where this template applies
- *Literal result elements* come from non-XSL namespace(s)
- XSLT elements come from the XSL namespace

XSLT Philosophien

- rekursiver Stil, gesteuert durch das Dokument

```
<xsl:template match="section/title">
  <h2><xsl:apply-templates/></h2>
</xsl:template>
```

- prozeduraler Stil, gesteuert durch Style Sheet

```
<xsl:for-each select="row">
  <tr><xsl:apply-templates/></tr>
</xsl:for-each>
```


XSLT Template Anwendung

- gesteuert durch das Laufzeitsystem
  - abstrakter als prozedurales Programmieren
  - mehr Kenntnisse über Laufzeitumgebung nötig
- in komplexen Style Sheets treten Konflikte auf
  - es treffen mehrere Patterns zu
  - manchmal Absicht, manchmal zufällig
- XSLT definiert Regeln für *Conflict Resolution*
  - Bewertung aller Templates nach festem Schema
  - explizite Beeinflussung der Priorität möglich
  - am höchsten bewertetes Template wird angewendet

XSLT Build-in Template Rules

- wichtig für das Verständnis des Ablaufes
  - im XSLT Standard definiert
  - werden häufig im Style Sheet überschrieben

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

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

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


Conditional Code in XSLT

- **<xsl:if>**: Simple conditional (no “else”)

```
<xsl:if test="{$somecondition}">
  <xsl:text>this text only gets used if $somecondition is
   true()</xsl:text>
</xsl:if>
```

- **<xsl:choose>**: Select among alternatives with **<xsl:when>** and **<xsl:otherwise>**

```
<xsl:choose>
  <xsl:when test="$count > 2"><xsl:text>, and
     </xsl:text></xsl:when>
  <xsl:when test="$count > 1"><xsl:text> and
     </xsl:text></xsl:when>
  <xsl:otherwise><xsl:text> </xsl:text></xsl:otherwise>
</xsl:choose>
```

Sortieren in XSLT (I)

```
<doc>
<para>Here's a table of sales:</para>
<table>
<row><cell>3000</cell><cell>Dreitausend</cell></row>
<row><cell>2400</cell><cell>zweite Zelle</cell></row>
<row><cell>10000</cell><cell>grosse Zahl</cell></row>
<row><cell>101</cell><cell>kleinste Zahl</cell></row>
</table>
</doc>
```


Sortieren in XSLT (II)

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

<xsl:template match="table">
  <xsl:if test="@id"><a name="{@id}"/></xsl:if>
    <table>
      <xsl:apply-templates select="row">
        <xsl:sort data-type="number" select="./cell[1]"/>
        do something...
      </xsl:apply-templates>
    </table>
</xsl:template>
</xsl:stylesheet>
```

Numerierung in XSLT

- **`<xsl:number>`** performs two functions:
  - it evaluates a numeric expression and converts the result into a formatted string:
    - `<xsl:number value="3" format="A. "/>`
    - `<xsl:number value="count(listitem)" format="01"/>`
  - it counts elements in the *source* tree and converts the result into a formatted string:
    - `<xsl:number count="listitem" format="i. "/>`
    - `<xsl:number count="chapter" from="book" level="any" format="1. "/>`
    - `<xsl:number count="h1|h2|h3" level="mulitple" from="chapter|appendix" format="1."/>`

Beispiel für XSL Formatting Objects

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/XSL/Format/1.0">
 ...
<fo:flow font-size="14pt" line-height="14pt">
<fo:block text-align="centered" font-size="24pt"
  line-height="28pt">
This is a simple XSL-FO document
</fo:block>
<fo:block space-before.optimum="12pt"
  text-align="centered">
by Erik Wilde
</fo:block> </fo:flow> </fo:root>
```

Zusammenfassung

- Style Sheet Languages für das Web
  - XSL für XML
- XSLT als generelle XML Transformation
  - extrem wichtig in XML-Umfeld
- Trend zum XML Content Management
  - Inhalte in XML erfassen und speichern
  - Publikation via HTML, XHTML, XML, WML, ...