Kapitel 7
Verändern von XML-Daten

- XML-Daten in Files:
  - üblicherweise keine Änderungen (ausser manuell mit Texteditoren/Interfaces)
  - Transformationen XML $\rightarrow$ HTML etc: XSLT
- XML-Daten in Anwendungssystemen
  - innerhalb der Anwendungsprogrammiersprache; meistens über DOM
  - keine Datenmanipulationssprache notwendig (vgl. OQL)?
- verschiedene Vorschläge
  - kommerzielles Umfeld:
    - XMLDB: XUpdate (2001; als Erweiterung zu XQuery)
    - eXcelon (2000; XUL als Erweiterung zu XSLT)
  - universitäres Umfeld:
    - "Updating XML" (Halevy et al, SIGMOD 2001) als Erweiterung zu XQuery
    - XPathLog (May, VLDB 2001): Prolog-artige Anfrage- und Manipulationssprache

Erweiterung von XQuery mit Updates – Konzepte

Inzwischen prinzipielle Klarheit, welche *Operationen* es geben soll. Syntax noch offen.

- immer ausgehend von einem Kontextknoten und ggf. Variablenbindungen
- anstelle der return-Klausel
- Basisoperationen:
  - delete *node*
  - rename *node* as *name*
  - insert *node/nodes* before|after|into *node*
- Kombinierte Operationen:
  - replace *node* with *node*
  - move *node* before|after|into *node*
- Benutzung derselben Notation (literales XML oder explizite Konstruktoren) wie in XQuery's return-Klausel.

XQUERY MIT UPDATES – BEISPIEL: QUIP

- Der Aufruf `quip filename.quip > bla.xml` schreibt das veränderte XML in ein File.

```
update
for $c IN document("/home/may/twocountries.xml")//country
let $area := string($c/@area)
delete $c/@area
insert <area>{$area}</area> after $c/name
rename $c//city[@id=$c/@capital] as capital
replace $c/@car_code with
    attribute code {concat($c/name/text(), ":", string($c/@car_code))}
replace $c/population/text() with $c/population/text() * $c/population_growth
insert "biggest city" into
   $c//city[population = max(for $citypop in $c//city/population/text()
                              return int($citypop))]
```


Kapitel 8
Die Transformationssprache XSL

8.1 XSL: Extensible Stylesheet Language

- Entstanden aus
  - CSS (Cascading Stylesheets) Skriptsprache zur Umsetzung von Datenquellen in HTML oder sonstiges optisches Markup, und
  - DSSSL (Document Style Semantics and Specification Language), Stylesheet-Sprache für SGML. Funktionale Programmiersprache.
- Prinzip: Beschreibt (regelbasiert) wie die Elemente *rekursiv* transformiert und formatiert werden:
  - Input: XML
  - Output: XML (Spezialfall: HTML)
- deklarativ/funktional: XSLT (XSL Transformations)

ANWENDUNGEN

- XML $\rightarrow$ XML
  - Transformation einer XML-Instanz in eine neue Instanz zu einer anderen DTD
  - Integration mehrerer XML-Instanzen zu einer
  - Extraktion von Teildaten aus einer XML-Instanz
  - Aufteilung einer XML-Instanz in mehrere Instanzen
- XML $\rightarrow$ HTML
  - Transformation einer XML-Instanz nach HTML zur Darstellung in einem Browser
- XML $\rightarrow$ beliebig
  - Da man keine Datenstrukturen, sondern nur ASCII erzeugt, kann man z.B. auch LaTeX, postscript, pdf generieren
  - ... oder nach XSL-FO (Formatting Objects) transformieren.

DIE SPRACHE(N) XSL

Aufteilung in zwei Teilsprachen:

- funktionale Programmiersprache: XSLT
  wird von *XSLT-Prozessoren* “verstanden” (z.B. xt, xalan, saxon, xsltproc ...)
- generische Sprache für Dokument-Markup: XSL-FO
  wird von XSL-FO-fähigen *Browsern* verstanden, die das XSL-FO-Markup anhand eigener Umsetzungsanweisungen in eine direkte (Bildschirm)darstellung umgesetzt werden.

- XSL ist selbst in XML-Syntax.
  Es benutzt die Namespaces “xsl:” und “fo:”
- Damit kann man XSL-Programme auch als XML-Daten auffassen.
- Prinzipiell kann man beliebige weitere Sprachen, die eine XML-Syntax (und einen eigenen Namensraum) haben, damit kombinieren.

ANWENDUNG: XSLT FÜR XML → HTML

- das zu verwendende Stylesheet wird in einer Processing Instruction im Prolog des XML-Dokuments angegeben:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="mondial-simple.xsl"?>
<!DOCTYPE mondial SYSTEM "mondial-2.0.dtd">
<mondial>
   ...
</mondial>
```

- Wenn ein (XSL-fähiger) Browser ein XML-Dokument mit Stylesheet-Angabe anzeigen soll, wird das XML-Dokument entsprechend dem Stylesheet verarbeitet, und das Ergebnis im Browser angezeigt.
- Hinweis: nicht alle Browser unterstützen die volle Funktionalität (id()-Funktion)
- I.a. hat man zu einer DTD ein passendes Stylesheet, seltener verwenden verschiedene Dokumente zur selben DTD verschiedene Stylesheets.

8.2 XSLT: Syntax und Semantik

- Jedes XSL-Stylesheet ist selber ein gültiges XML-Dokument,

```
<?xml version="1.0">
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   ...
</xsl:stylesheet>
```

  - enthält Elemente aus dem Namespace xsl:, welche die Transformation/Formatierung beschreiben.
  - enthält literales XML, um das Ergebnisdokument zu erzeugen
  - verwendet XPath-Ausdrücke zum Zugriff auf Knoten ein einem XML-Dokument. XPath-Ausdrücke stehen (meistens) als Attribute von <xsl:...>-Elementen.
  - XSL-Stylesheets/Programme erzeugen rekursiv einen Result-Tree aus einem XML-Input-Tree.

8.2.1 XSLT: Ablaufsteuerung durch Templates

Das Stylesheet besteht hauptsächlich aus *Templates*, die angeben, welche Elemente wie bearbeitet werden sollen:

- xsl:template:

  <xsl:template match="*xsl-pattern*"
  content
  </xsl:template>

- *xsl-pattern* ist ein XPath-Ausdruck, der ohne Verwendung von "*axis*::" gebildet ist (vgl. Folie 174). Er gibt an, auf welche Elemente das Template *anwendbar ist*:
  Ein Knoten x erfüllt *xsl-pattern*, wenn es irgendeinen Ancestor-Knoten k von x gibt, so dass x in der Ergebnismenge von *xsl-pattern* für k als Kontextknoten enthalten ist.

  (eine weitere Selektion findet später statt, wenn entschieden wird auf welche Elemente "ihre" Templates angewendet werden sollen).

- *content* enthält die Anweisungen zur Erzeugung eines Fragments des Result-Trees.

TEMPLATES

- <xsl:template match="city">
  <xsl:copy-of select="current()"
  </xsl:template>

  ist ein Template, das auf Städte angewendet werden kann, und diese unverändert in den Ergebnis-Baum kopiert.

- <xsl:template match="lake|river|sea"> ... </xsl:template>
  kann auf Gewässer angewendet werden.

- <xsl:template match="country/province/city"> ... </xsl:template>
  kann auf city-Elemente, die Subelemente von province-Elementen, die wiederum Subelemente von country-Elementen sind, angewendet werden.

- <xsl:template match="id('D')"> ... </xsl:template>
  kann auf das Element, dessen ID "D" ist, angewendet werden.

- <xsl:template match="city[population > 1000000]"> ... </xsl:template>
  kann auf city-Elemente, die mehr als 1000000 Einwohner haben, angewendet werden.

TEMPLATES

- xsl:apply-templates:

  <xsl:apply-templates select=“*xpath-expr*”/>

- *xpath-expr* ist ein XPath-Ausdruck, der angibt, auf welche Elemente (ausgehend von dem Knoten, auf den das aktuelle Template angewendet wird als Kontextknoten) das jeweils für sie definierte Template angewendet werden soll.
- Mit <xsl:apply-templates-Elementen> innerhalb des Contents von <xsl:template>-Elementen wird die hierarchische Stuktur von XML-Dokumenten verarbeitet (und ggf. tiefgreifend verändert).
- Will man alle Subelemente abbilden, kann das select-Attribut weggelassen werden.

  <xsl:apply-templates/>

TEMPLATES

- <xsl:apply-templates select=“country”/>
  behandelt alle country-Subelemente des aktuellen Kontextelements
- <xsl:apply-templates select=“country/city”/>
  behandelt alle city-Subelemente von country-Subelementen des aktuellen Kontextelements,
- <xsl:apply-templates select=“/mondial//city[population > 1000000]”/>
  behandelt alle city-Elemente, die Mondial enthalten sind, und deren Population > 1000000 ist,
- <xsl:apply-templates select=“id(@capital)”/>
  behandelt das Element, dessen ID gleich dem Wert des capital-(Referenz)attributs des aktuellen Kontextelements ist.

TEMPLATES

- Es muss ein Template geben, das das root-Element verarbeiten kann, damit der Verarbeitungsprozess anläuft:
  - <xsl:template match="*name_des_root_elementes*">
  - <xsl:template match="/">
  - <xsl:template match="*">

KONFLIKTE BEIM AUFRUF VON TEMPLATES

- Sind bei der Abarbeitung einer <xsl:apply-templates>-Anweisung mehrere Templates anwendbar, wird das mit der speziellsten match-Spezifikation aufgerufen.

TEMPLATES: BEISPIEL

mondial-simple.xsl: Darstellung der countries als Tabelle (→ HTML)

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

 <xsl:template match="mondial">
   <html> <body> <table>
            <xsl:apply-templates select="country"/>
       </table>  </body> </html>
 </xsl:template>

 <xsl:template match="country">
   <tr><td> <xsl:value-of select="name"/> </td>
       <td> <xsl:value-of select="@car_code"/> </td>
       <td align="right"> <xsl:value-of select="population"/> </td>
       <td align="right"> <xsl:value-of select="@area"/> </td>
   </tr>
 </xsl:template>
</xsl:stylesheet>
```

Templates: Beispiel

Das folgende Stylesheet kopiert alle country und city-Elemente aus Mondial und gibt erst alle country, dann alle city-Elemente als top-Level-Elemente aus:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <!-- Template definieren -->
   <xsl:template match="city|country">
      <xsl:copy-of select="current()"
   </xsl:template>
   <template match="mondial">
      <!-- Template anwenden: erst Countries -->
      <xsl:apply-templates select="/mondial/country">
      <!-- Template anwenden: dann Cities -->
      <xsl:apply-templates select="//country/city | //country/province/city"/>
   </ template>
</xsl:stylesheet>
```


Templates

Unterschied zwischen:

1. <xsl:template match="*xsl-pattern*"
      content
   </xsl:template>
2. <xsl:apply-templates select="*xpath-expr*"/>

- select="..." wird bezüglich dem aktuellen Kontextknoten ausgewertet (wird das Element durch den gegebenen XPath-Ausdruck adressiert?),
- match="..." wird bezüglich der allgemeinen Dokumentstruktur ausgewertet ("matcht" die Dokumentstruktur mit dem gegebenen Pattern?),
- xsl:apply-templates wendet auf alle durch *xpath-expr* adressierten Knoten (der Dokumentordnung entsprechend) das passende Template an,
- die Reihenfolge der Templates hat keinen Einfluß auf die Anwendungsreihenfolge.

Templates

Aufgabe 6
Was ist der Unterschied zwischen den folgenden Stylesheet-Fragmenten?

1. ```
   <xsl:template match="city">
       <xsl:copy-of select="current()"
   </xsl:template>
   <xsl:apply-templates select="//country/city"/>
   <xsl:apply-templates select="//country/province/city"/>
   ```

2. ```
   <xsl:template match="country/city">
       <xsl:copy-of select="current()"
   </xsl:template>
   <xsl:template match="country/province/city">
       <xsl:copy-of select="current()"
   </xsl:template>
   <xsl:apply-templates
       select="//country/city|//country/province/city">
   ```

□

Benannte Templates

Es gibt auch benannte Templates, die man aufrufen kann (vgl. Makros)

- xsl:template mit name-Attribut:

  ```
  <xsl:template name="name">
      content
  </xsl:template>
  ```

  – *name* ist ein beliebig gewählter Name
  – *content* kann xsl-Statements enthalten, z.B. xsl:value-of, die bezüglich dem jeweils aktuellen Kontextknoten ausgewertet werden.

- xsl:call-template

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

8.2.2 XQuery und XSLT

- beides deklarative, funktionale Sprachen ...
- ... mit komplett unterschiedlichen Strategien:
  - XQuery: Schachtelung der Anfragesyntax entspricht direkt der Struktur des Ergebnisses
  - XSLT: erst durch die geschachtelte Abarbeitung der Templates ergibt sich die Struktur des Ergebnisses.

XSLT

- sehr modularer Aufbau der Stylesheets
- Erweiterbarkeit, Wiederverwendung von Templates
- sehr flexible datengesteuerte Auswertung innerhalb des Aufrufbaumes

XQuery

- klare Vorteile bei Joins (for $a in ..., $b in ...)
- hier muss in XSLT das Join explizit ausprogrammiert werden (xsl:for-each)

UMSETZUNG XSLT → XQUERY

- Jedes Template wird als FLWR-Statement umgesetzt,
- innere Template-Aufrufe ergeben geschachtelte FLWR-Statements innerhalb der return-Klausel
- Allgemeinheit von z.B. <apply-templates/> kann nicht direkt abgedeckt werden, da man nicht weiss, welches Template aufgerufen wird.

⇒ je flexibler das Schema (Dokumente) desto mehr ist XSLT gegenüber XQuery überlegen.

Aufgabe 7

- Geben Sie XQuery-Anfragen an , die dasselbe tun, wie mondial-simple.xsl bzw. mondial-nested.xsl
- Geben Sie eine XQuery-Anfrage an, die dasselbe tut wie das Stylesheet auf Folie 248. □

8.2.3 XSLT: Erzeugung des Ergebnisbaumes

In den Ergebnisbaum können auf verschiedene Arten Knoten eingefügt werden:

- Textuelles Schreiben von Werten und Attributen,
- Kopieren von Knoten und Werten aus dem Eingabe-Baum,
- Erzeugen von Elementen und Attributen durch Konstruktoren.

Textuelles Erzeugen von Struktur und Inhalt

- Alle Tags, komplette Elemente und Attribute im Template-*content*, die nicht zum xsl-namespace (oder zum lokalen Namespace des verwendeten Tools) gehören, werden in den Result-Tree eingefügt.
- mit <xsl:text> *some_text*</xsl:text> kann explizit Text eingefügt werden (whitespace, z.B. auch bei der Generierung von IDREFS-Attributen).

Erzeugung des Ergebnisbaumes

Kopieren aus dem Eingabe-Baum

- <xsl:copy/>
  kopiert den aktuellen Kontextknoten (bzw. “seine Hülle”): alle zu ihm gehörenden Namespace-Knoten, aber *nicht* seine Attribute und Subelemente.
- <xsl:copy-of select=“*xpath-expr*”/>
  kopiert das Ergebnis von *xpath-expr* (angewendet auf den aktuellen Kontext) unverändert in den Result-Tree.
- <xsl:value-of select=“*xpath-expr*”/>
  erzeugt einen Text-Knoten mit dem Wert von *xpath-expr*.

Aufgabe 8
Machen Sie sich den Unterschied zwischen <xsl:copy/>, <xsl:copy-of select=“current()”/> und <xsl:value-of select=“current()”/> klar.
In welchen Fällen haben je zwei der Kommandos dasselbe Ergebnis? □

Erzeugung des Ergebnisbaumes

Beispiel:

```
<xsl:template match="city">
  <mycity>
    <xsl:value-of select="name"/>
    <xsl:copy-of select="longitude|latitude"/>
  </mycity>
</xsl:template>
```

- erzeugt für jedes city-Elemente ein mycity-Element,
- der Name wird als #PCDATA-Contents aufgenommen,
- die Subelemente longitude und latitude werden kopiert.

Erzeugung des Ergebnisbaumes

Um Attributwerte zu setzen, kann

```
<xsl:value-of select="xpath-expr"/>
```

nicht *direkt* verwendet werden. An dieser Stelle können stattdessen XPath-Ausdrücke in {...} eingeschlossen werden:

```
<xsl:template match="city">
  <mycity id="{@id}">
    <xsl:value-of select="name"/>
    <xsl:copy-of select="longitude|latitude"/>
  </mycity>
</xsl:template>
```

XSLT: Erzeugung des Ergebnisbaumes

Beispiel:

```
<xsl:template match="city">
  <mycity source="mondial"
          country="{ancestor::country/name}">
    <xsl:apply-templates/>
  </mycity>
</xsl:template>
```

- erzeugt für jedes city-Elemente ein mycity-Element,
- konstantes Attribute "source",
- Attribut country, das das Land, in dem die Stadt liegt, angibt,
- alle anderen Attribute fallen weg,
- auf alle Subelemente werden entsprechende Templates angewendet.

XSLT: Erzeugung des Ergebnisbaumes

Erzeugen von Elementen und Attributen

- <xsl:element name="*xpath-expr*">
  content
  </xsl:element>
  erzeugt ein Element vom Elementtyp *xpath-expr* im Result-Tree, der Inhalt des neuen Elements ist *content*. Hiermit können Elementnamen neu berechnet werden.
- <xsl:attribute name="*xpath-expr*">
  content
  </xsl:attribute>
  erzeugt ein Attribut mit Namen *xpath-expr* und Wert *content* für das momentan in Konstruktion befindliche Element im Result-Tree (entweder textuell oder mit xsl:element erzeugt).
  Hiermit können Attributnamen neu berechnet werden.
- Mit <xsl:attribute-set name="*name*"> xsl-attribute* </xsl:attribute-set>
  können Attributmengen vordefiniert und von xsl:element und xsl:copy via
  use-attribute-sets="attr-set$_1$ ... attr-set$_n$" verwendet werden.

8.2.4 XSLT: Kontrollstrukturen

- Templates: rekursive Steuerung des Ablaufes
- weitere Kontrollstrukturen innerhalb des Inhaltes der Templates:
  - Schleifen
  - Verzweigungen

ITERATIONEN

Sind viele gleichartige Subelemente oder ein mehrwertiges Attribut zu verarbeiten, kann man eine lokale Iteration ausführen:

```
<xsl:for-each select="xpath-expr">
    content
</xsl:for-each>
```

- eine xsl:for-each-Iteration ist u.a. notwendig, wenn etwas anderes als das Verhalten des für die betreffenden Elemente definierten Templates durchgeführt werden soll.

ITERATIONEN: BEISPIEL

- XML-Source: "border"-Subelemente von "country" mit IDREF-Attribut "country":
  <border country="*car_code*" length="...">
- Result-Tree: Erzeugung eines IDREFS-Attributes country/@neighbors, das alle Nachbarländer enthält (Benutzung von <xsl:text>).

```
<xsl:template match="country">
  <xsl:element name="country">
    <xsl:copy-of select="attribute::*"/>
    <xsl:attribute name="neighbors">
      <xsl:for-each select="border">
        <xsl:value-of select="@country"/>
        <xsl:text> </xsl:text>
      </xsl:for-each>
    </xsl:attribute>
  </xsl:element>
</xsl:template>
```

XSLT: Bedingte Verarbeitung

- Einfacher Test:
  <xsl:if test="*predicate*"> *content* </xsl:if>

Beispiel:

```
<xsl:template match="country">
  <table>
    <tr> <th colspan="2"> <xsl:value-of select="name"> </th>
    </tr>
    <xsl:if test="@area">
      <tr>
        <td> Fläche: </td>
        <td> <xsl:value-of select="@area"> </td>
      </tr>
    </xsl:if>
    ...
  </table>
</xsl:template>
```


XSLT: Bedingte Verarbeitung

- Mehrere Alternativen:

```
<xsl:choose>
   <xsl:when test="predicate_1">
      contents_1
   </xsl:when>
   <xsl:when test="predicate_2">
      contents_2
   </xsl:when>
   ...
   <xsl:otherwise>
      contents_n+1
   </xsl:otherwise>
</xsl:choose>
```

8.2.5 XSLT: Variablen und Parameter

Variablen und Parameter dienen dazu, Bezeichner an Werte zu binden.

Variablen

- Variablen können nur einmal (bei Definition) einen Wert zugewiesen bekommen.
- eine Variable kann nur an einmal definiert werden und einen Wert zugewiesen bekommen; eine Neuzuweisung ist später nicht mehr möglich.
- Variablen können als top-level-Elemente definiert sein. Dann sind sie im gesamten Stylesheet sichtbar ("Konstante").
- Eine Variablendefinition kann an beliebiger Stelle innerhalb eines Templates stehen, eine solche Variable, ist in ihren nachfolgenden Geschwistern sichtbar
  - also ggf. z.B. nur innerhalb eines <xsl:for-each>,
  - dann bekommt sie bei jedem Durchlauf einen neuen Wert zugewiesen.

PARAMETER

Parameter

- ... ähnlich wie Variablen
- Will man Werte an aufgerufene Templates übergeben, geschieht dies durch Parameter.
- Die Definition von Parametern ist *nur* am Anfang von xsl:template-Elementen erlaubt. Der definierte Parameter ist dann überall in diesem Template sichtbar.
- Die Wertzuweisung geschieht im allgemeinen innerhalb des aufrufenden <xsl:apply-templates>- oder <xsl:call-template>-Elements durch Angabe des zu übergebenden Wertes.
- reiner Wertparameter (call-by-value, kein call-by-reference möglich).

Hinweis: da der Parameter ein ganzes Element sein kann, reicht theoretisch in einziger Parameter immer aus.

VARIABLEN UND PARAMETER

- Wertzuweisung entweder durch “select”-Attribut (Wert ist ein String, ein Knoten, oder eine Menge von Knoten)

  <xsl:variable name=“*var-name*” select=“*xpath-expr*”/>

- oder durch Element-Contents (damit kann der Wert einer Variablen ein im XSLT-Skript erzeugter Baum sein):

  <xsl:variable name=“*var-name*”>
  content
  </xsl:variable>

- Analog für xsl:param.
- Verwendung: $*var-name*

PARAMETERÜBERGABE AN TEMPLATES

- Parameter werden am Beginn von Templates definiert:

  <xsl:template match=“...”>
  <xsl:param name=“*param-name*”
  select=“*xpath-expr/*”> <!-- mit Angabe eines Default-Wertes -->
  ⋮
  </xsl:template>

- Beim Aufruf des Templates werden dann Parameterwerte angegeben

  <xsl:apply-templates select=“*xpath-expr*$_1$”>
  <xsl:with-param name=“*param-name*”
  select=“*xpath-expr*$_2$”/>
  </xsl:apply-templates>

BEISPIEL

Es soll eine Liste erstellt werden, die alle Organisationen mit jedem ihrer Mitglieder auflistet. Dazu wird die Abkürzung der Organisation als Parameter an das country-Template übergeben, das dann den Eintrag erzeugt.

$\rightarrow$ nächste Folie

[Filename: orgs-and-members.xsl]

BEISPIEL (FORTS.)

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="mondial">
    <html><body> <h2>Mitgliedschaften-Tabelle</h2>
      <table>
        <xsl:apply-templates select="organization"/>
      </table></body></html>
  </xsl:template>
  <xsl:template match="organization">
    <tr><td colspan="2"><xsl:value-of select="name"/></td></tr>
    <xsl:apply-templates select="id(members/@country)">
       <xsl:with-param name="the_org" select="name/text()"/>
    </xsl:apply-templates>
  </xsl:template>
  <xsl:template match="country">
    <xsl:param name="the_org"/>
    <tr><td><xsl:value-of select="$the_org"/></td>
        <td><xsl:value-of select="name/text()"/></td></tr>
  </xsl:template>
</xsl:stylesheet>
```

Aufgaben

Aufgabe 9

- Erweitern Sie das obige Stylesheet so, dass auch die Art der Mitgliedschaft ausgegeben wird.
- Schreiben Sie ein äquivalentes Stylesheet, das keinen Template-Aufruf durchführt, sondern explizit mit <xsl:for-each> funktioniert.
- Geben Sie eine äquivalente XQuery-Anfrage an. □

Aufgabe 10

Erstellen Sie ein Stylesheet zu der von Ihnen gewählten Anwendung um Ihr(e) XML-Instanz(en) im Web darzustellen. □

8.2.6 XSLT: Sonstiges

Sortieren

Bei den mengenbasierten XSLT-Elementen

- xsl:apply-templates und
- xsl:for-each

kann angegeben werden, nach welchen Schlüsseln sortiert werden soll:

<xsl:sort select="*xpath-expr*"
    data-type = {"text"|"number"}
    order = {"descending"|"ascending"}/>

- select spezifiziert den Schlüssel, nach dem geordnet werden soll,
- data-type gibt an, ob dieser Wert als Text oder als Number interpretiert werden soll,
- order gibt an ob auf- oder absteigend geordnet werden soll,
- hat ein xsl:apply-templates oder xsl:for-each-Element mehrere xsl:sort-Subelemente, werden diese geschachtelt angewendet (wie in SQL).

IMPORT-MECHANISMEN

XSL-Stylesheets können weitere Stylesheets importieren (d.h., man kann die dortigen Regeln importieren):

- <xsl:include href="*url*"/>
  für konfliktfreie Stylesheets,
- <xsl:import href="*url*"/>
  Definitionen des importierenden Dokuments haben höhere Priorität als Definitionen des importierten Dokuments,
  Die xsl:import-Subelemente müssen vor allen anderen Subelementen stehen.

EINLESEN VON DATEN AUS MEHREREN DOKUMENTEN

- mit der document()-Funktion aus XPath:

```
<xsl:template match="*">
 <html>
   <xsl:copy-of select="document('file:/home/may/paper.html')//h5"/>
 </html>
</xsl:template>
```

ERZEUGEN MEHRERER INSTANZEN

... abhängig von der verwendeten Software

saxon: <xsl:document href="*ausgabefile*">

XSLT-2.0: <xsl:result-document href="*ausgabefile*">

```
<?xml version="1.0" encoding="utf-8"?>
<!-- xsl:document is saxon/xslt1.1 for xslt2.0's xsl:result-document -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.1">
 <xsl:template match="*">
  <xsl:document href="countries.xml">
    <countries><xsl:apply-templates select="country"/></countries>
  </xsl:document>
  <xsl:document href="organizations.xml">
    <organizations><xsl:apply-templates select="organization"/></organizations>
  </xsl:document>
 </xsl:template>
 <xsl:template match="country"><xsl:copy-of select="name"/></xsl:template>
 <xsl:template match="organization"><xsl:copy-of select="name"/></xsl:template>
</xsl:stylesheet>
```

[Filename: redirected-output.xsl]

ERWEITERUNGSFUNKTIONEN

- Unterschiedliche Implementierungen definieren seit der Anfangszeit der Entwicklung von XSLT 1.0 immer wieder eigene "Extension Functions"
- einige davon in XSLT 2.0 aufgenommen
- einiges immer noch offen
- EXSLT, siehe `http://www.exslt.org` koordiniert weitere Ergänzungsvorschläge
  - Benutzerdefinierte Funktionen
  - Datum, Zeit; Stringfunktionen
  - reguläre Ausdrücke
  - "Dynamic": Auswertung generierter XPath-Ausdrücke
    (vorher z.B. `saxon:evaluate`)
    (vgl. Dynamic SQL package für Oracle)

8.3 XSL-FO

XSL-FO spezifiziert *Formatting-Objects*, die mit dem XSLT-Skript in den Result-Tree eingebaut werden und gezielt die spätere Formatierung beschreiben, z.B.

- Seitenlayout, Bereiche, Rahmen, Einrückung,
- Farben, Schriftarten und -größen,
- Strukturierung, z.B. Listen, Tabellen ...

Es ist so ziemlich alles vorgesehen, was man z.B. von LaTeX kennt.

FO-Objekte sind z.B. (Namespace fo:) fo:block, fo:character, display-graphic, float, footnote, inline-graphic, list-block, list-item, list-item-body, list-item-label, multi-case, page-number, page-number-citation, region-before/after, region-body, simple-link, table, table-and-caption, table-body, table-caption, table-cell, table-column, table-footer, table-header, table-row.

- Jedes dieser Objekte kann entsprechende Attribute haben.

XSL-FO

- Result-Tree enthält *Formatting Objects*-Elemente
- ist dann Eingabe für Formatter, der daraus HTML/LaTeX/RTF/PDF etc. erstellt,
- (derzeit einziges?) Tool, das XSL-FO versteht:

  FOP (by James Tauber) , ein Java-Programm, das XML-Dokumente, die XSL-FO-Markup enthalten, nach PDF übersetzt:

  `http://www.jtauber.com/fop/`

8.4 XSLT: Sprachdesign in der XML-Welt

- XSLT ist selber in XML-Syntax
- Es gibt eine DTD dazu: `http://www.w3.org/TR/xslt#dtd`

⇒ Analog gibt es auch eine XML-Syntax für XQuery: XQueryX, `http://www.w3.org/TR/xqueryx`.

- XSLT benutzt einen eigenen *Namespace*, xsl:....
- Es gibt viele weitere Sprachen dieser Art (Programmiersprachen, Auszeichnungssprachen, Repräsentationssprachen ...):
  XLink, XML Schema,
  SOAP (Simple Object Access Protocol)
  WSDL (Web Services Description Language)
  OWL (Web Ontology Language)
  DocBook
  ... jede Menge anwendungsspezifische Sprachen.