XSLT, XSL-FO
Transformieren und Formatieren

http://www.w3.org/TR/xslt,
http://www.w3.org/TR/xsl

Übersicht zu XSL

XSL: Extensible Stylesheet Language

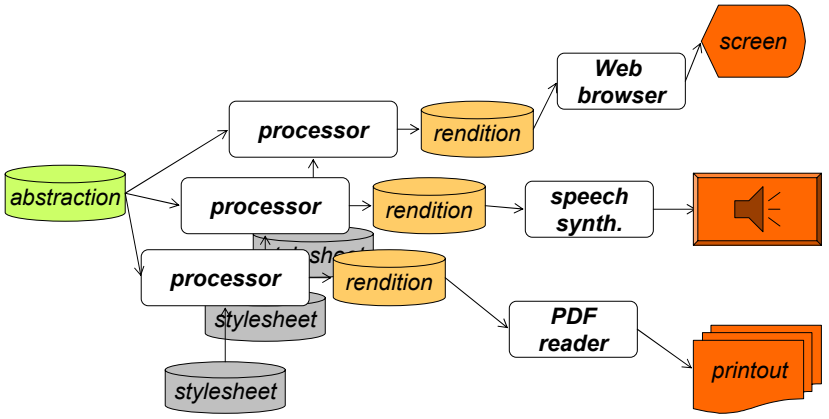
- Ziel:
  - Darstellung eines XML-Dokuments (Erzeugung von „*renditions*“)
- Historische Vorläufer:
  - **DSSSL** (*Document Style Semantics and Specification Language*), der Standardweg zur Verarbeitung/Anzeige von SGML-Dokumenten)
  - **CSS2** (*Cascading Stylesheets level* 2), primär zur Layoutkontrolle von HTML-Seiten.
- Ansatz:
  - Schaffung einer XML-basierten Beschreibungssprache für die Darstellung auf Ausgabemedien wie Papierseiten, „scrollbare“ Fenster, kleine PDA-Displays oder Sprachsynthesizer,
  - Die Formatierungssemantik wird ausgedrückt als eine Art Bibliothek bestimmter Objektklassen, der ***Formatting Objects***.

XSL: Extensible Stylesheet Language

- Teil-Technologien:
  - XSL Transformations (XSLT)
    - Aus dem XML-Quelldokument wird ein XML-Zieldokument gewonnen.
    - Dies geschieht durch Transformation, d.h. die Konstruktion eines neuen Dokumentbaums aus dem alten.
    - Diese Transformation von XML-Dokumenten erwies sich als eigenständige Aufgabe, die auch unabhängig von Formatierungen ihren Wert besitzt, und wurde daher als eigene Spezifikation formuliert.
  - Formatting Objects (FO)
    - Die Spezifikation zu FO bildet den eigentlichen Kern von XSL.
    - Hier wird die (XML-basierte) Beschreibungssprache für die Präsentation von Daten definiert.
    - XSL-FO ist komplex und umfangreich. Zu Verständnis ist erhebliches Hintergrundwissen zu allgemeinen Darstellungsfragen erforderlich.

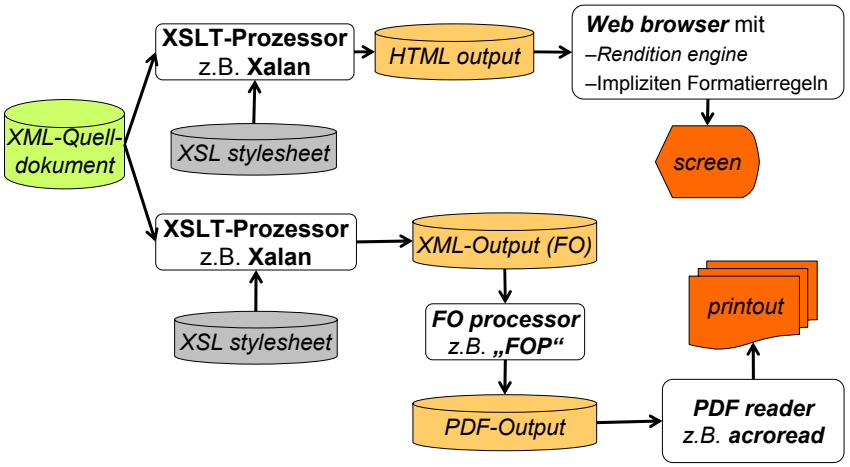
XSL

Erinnerung: Von der *abstraction* zur *rendition*, allgemein…



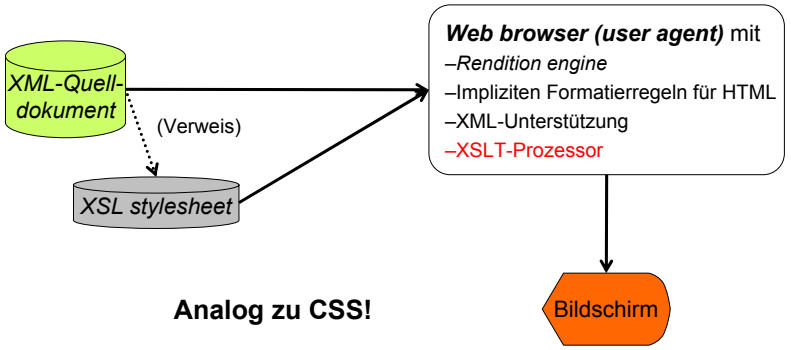
XSL

… und nun konkreter:



XSL

… oder auch:

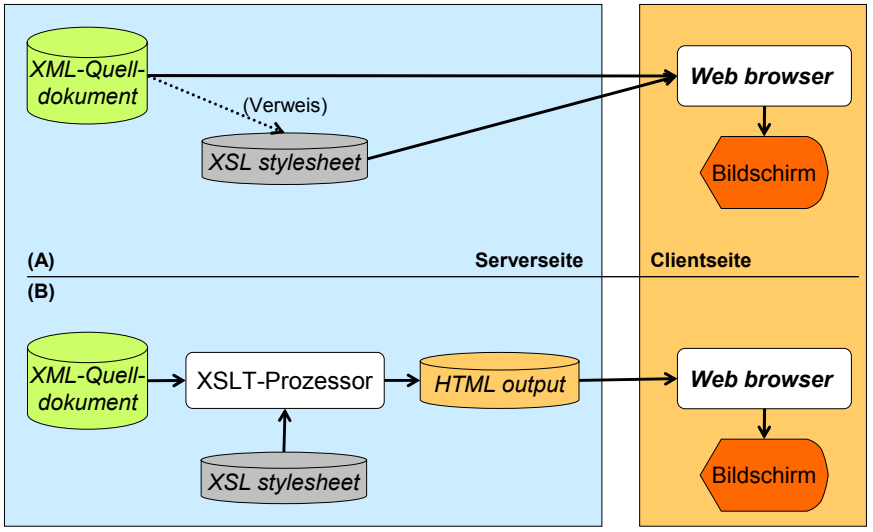


Analog zu CSS!

XSLT

- Demo
  - „Othello“-Szene
    - („Wrox“-Buchbeispiel aus Kap.8)
  - mit XML Spy und IE am PC

XSL: Client/Server-Aspekte



XSL: Client/Server-Aspekte

- Vorteile der Client-seitigen Transformation:
  - Entlastung des Servers durch Verteilung der Transformationslast
  - Zugriff des Clients auf die originalen XML-Dokumente mit dem vollen Informationsgehalt der *abstraction*-Ebene
    - Beispiel Bestelldaten, EDI-Kontext:
      Neben Visualisierung auch Datenübernahme möglich
  - Caching der Stylesheets möglich
- Vorteile der Client-seitigen Transformation:
  - Geringe Anforderungen an den Client
  - Auswahl der übermittelten Information leichter möglich

XSLT

XSLT: Aufbau

- Aufbau der Sprache
  - XSL *stylesheets* verwenden XML-Syntax
    - ähnlich wie XML Schema
    - aber im Unterschied zu CSS
  - Elementtypen der Transformationssprache werden von den zu generierenden Elementtypen (z.B. XML-Zielformat, HTML, XSL-FO) mittels Namensräumen / Prefix-Angaben unterschieden, analog zu XML Schema
- Status
  - 1999-11-16: Version 1.0 (*recommendation*)
    - Autor: James Clark
  - 2002-11-15: Version 2.0 (4. Entwurf, noch nicht offiziell)
    - Verwendet XPath 2.0 und unterstützt XML Schema
    - Autor: Michael Kay, Software AG

XSLT: Ausgabe

- Ausgabeformate
  - „xml“: Zielformat ist XML
    - Der Normalfall, typisch für FO.
  - „html“: Zielformat ist HTML
    - unterscheide XHTML – das ist eine XML-Ausprägung
    - Dieser Modus wird nicht von allen XSLT-Prozessoren unterstützt. Er bewirkt z.B. die Vermeidung der XML-typischen *empty elements* wie `<foo/>`. HTML-eigene derartige Elemente werden ohne Ende-*tag* generiert, etwa `<br>`.
  - „text“: Zielformat ist normaler Text
    - Der Prozessor schreibt den Stringwert des jeweiligen Knotens heraus, ohne weitere Formatierung.
    - Nicht XML-konforme konstante Texte sind hier zulässig.

XSLT: Vorgehensweise

- XSLT allein kann ein ganzes Buch füllen
  - Z.B. „XSLT Programmer's Reference“ von Michael Kay, Wrox Press, 2000.
  - Eine erschöpfende Behandlung dieses Themas wird von der Stofffülle in diesem Rahmen ausgeschlossen.
- Daher nun induktives Vorgehen:
  - Vorstellen – und sofortiges Nachvollziehen am Rechner – einiger Code-Beispiele (Mischung Vorlesung & Übung)
  - Klärung dabei auftauchender konzeptioneller Fragen
  - Lösung konkreter kleiner Aufgaben
  - Dabei Aufbau eines kleinen Repertoirs der XSLT-Möglichkeiten
  - Nachlesen weiterer Möglichkeiten und ausgelassener Angaben, Einschränkunden, usw. in den Spezifikationen!

XSLT: Vorbereitung

- Übung 13 (im Rahmen der Vorlesung)
  - Legen Sie ein Unterverzeichnis „ueb13“ an, analog „ueb10“.
  - Kopieren Sie „ueb13.xml“ und „ueb13.xsd“ vom entsprechenden Verzeichnis des Dozenten dorthin.
    - Es handelt sich um leicht erweiterte Varianten von ueb10d.
  - Legen Sie im folgenden die Stylesheet-Dateien unter dem Namen „ueb13*x*.xsl“ an, mit *x*=(a, b, c, …)
  - Verwenden Sie folgende Aufrufe des XSLT-Prozessors:

```
Xalan src.xml sheet.xsl  # Ausgabe nach stdout
Xalan src.xml sheet.xsl > out.xml
```

  - Hinweise:
    - Dabei ersetzen Sie die Platzhalternamen durch die aktuellen
    - Xalan ist der XSLT-Prozessor der Apache Foundation und verwendet xerces als XML Prozessor.
    - Ggf. ergänzen Sie den vollständigen Pfad vor Xalan oder legen ein geeignetes alias in Ihrer Datei ~/.bashrc an.

XSLT: Leeres Stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl=
  "http://www.w3.org/1999/XSL/Transform"
  version="1.0">
<xsl:output method="xml"/>
   <!-- Regelsammlung der Schablonen hier -->
</xsl:stylesheet>
```

Aufgabe:

- Legen Sie die o.g. Eingabe als Datei ueb13a.xsl an.
- Vergleichen Sie die Ausgabe von Xalan für
  - method="xml",
  - method="html" und
  - method="text" sowie zur Prüfung des XSLT-Prozessors
  - method="test"

XSLT: Leeres Stylesheet

- Beobachtungen:
  - Im Fall „xml“ erscheint die XML-Deklaration zusätzlich, sonst sind die Outputs gleich.
  - Der Fall „test“ führt zu einer Xalan-Fehlermeldung.
  - Es erscheint Output (offenbar der „Textinhalt“ des Dokuments)
- Resultierende Frage
  - Woher stammt der Output, obwohl keine Regel hinterlegt ist?
- Dazu erst ein wenig Hintergrund-Information:

XSLT: Deklaratives Paradigma

- Die Sprache ist deklarativ
  - Ein *stylesheet* besteht i.w. aus einer Regelsammlung von Schablonen (*template rules*).
  - Die Regeln sind unabhängig voneinander und konzentrieren sich auf das, „was“ geschehen soll.
  - Die Frage „wie“ (z.B. Reihenfolge, Datenquellen, Verwaltung temporären Arbeitsspeichers etc.) bleibt dem XSLT-Prozessor überlassen!
  - Häufigster Fehler:
    - Verwirrung durch Denken im imperativen Paradigma der Programmierung!
  - Also: C, C++, Java, Perl hier vergessen und an SQL denken.

XSLT: Schablonenregeln

- Gliederung einer Schablonenregel
  - Abgebildet durch Elementtyp `<xsl:template>`
  - (Such-)Muster (*pattern*)
    - Definiert die Knotenmenge des Quelldokuments, auf die die Schablone angewendet werden soll.
    - Abgebildet durch Attribut `match`
    - Die Attributwerte sind i.w. XPath-Ausdrücke
  - Schablone (*template*)
    - Die eigentliche Anweisung, was mit den gefundenen Knoten geschehen soll.
    - Abgebildet schlicht als Elementinhalt.
  - Unterscheide „Schablone“ von „Schablonenregel“!

XSLT: Schablonenregeln

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl=
   "http://www.w3.org/1999/XSL/Transform"
   version="1.0">
<xsl:output method="html"/>
    <xsl:template match="/">
      <html><body>
      <h1>Hello</h1>
      <p>Hello World!<br/></p>
      </body></html>
    </xsl:template>
</xsl:stylesheet>
```

- ueb13b.xsl:
  - Eine einfache Schablonenregel
  - zugleich ein Beispiel für HTML-Erzeugung

XSLT: Schablonenregeln

- Beobachtungen:
  - Leiten Sie die Ausgabe in eine Datei ueb13b.html
  - Öffnen Sie diese mit Ihrem Browser. Ist die Anzeige ok?
  - Betrachten Sie den HTML-Code im *stylesheet* und in der Ausgabe.
    - Was fällt Ihnen am Element
 auf?
    - Was passiert, wenn Sie die Output-Syntax bereits im Stylesheet verwenden?
- Einschränkungen
  - Bisher nur Ausgabe statischer Angaben – wieso dann „Schablone“?
  - Wie erfolgt der Umgang mit anderen Knoten?

XSLT: Implizite Regeln

- Es gibt eingebaute (implizite) Regeln!
  - Ursache des Xalan-Outputs bei Beispiel A trotz Fehlens jeglicher Schablonenregeln!
- Priorisierung:
  - Analog zu „importierten“ Regeln
    - Hinweis: <xsl:import>
  - Interne Regeln haben Vorrang!
    - Konsequenz: Überladen der eingebauten Regeln deaktivieren sie.

XSLT: Implizite Regeln

- `<xsl:template match="*|/"> <xsl:apply-templates/> </xsl:template>`
  - Selektiert den root-Knoten und alle Elementknoten.
  - `<xsl:apply-templates>` ruft Schablonenregeln für die selektierten Knoten auf – aus einer solchen Regel heraus.
- `<xsl:template match="*|/" mode="m"> <xsl:apply-templates mode="m"/> </xsl:template>`
  - Analog, für jede Einschränkung mittels `mode`-Attribut (vgl. Kap. 5.7)
- `<xsl:template match="text()|@*"> <xsl:value-of select="."/> </xsl:template>`
  - Selektiert alle Text- und Attributknoten
  - `<xsl:value-of>` gibt den Stringwert des Kontextknotens aus
  - Diese Regel verursachte unsere Outputs!
- `<xsl:template match="processing-instruction()|comment()"/>`
  - Selektiert alle PI- und Kommentarknoten. Keine Ausgabe!

XSLT: Implizite Regeln

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl=
  "http://www.w3.org/1999/XSL/Transform"
  version="1.0">
<xsl:output method="xml"/>
   <xsl:template match="text() | @*">
   <!-- Keine Schablone – ignoriere Knoten -->
   </xsl:template>
</xsl:stylesheet>
```

- ueb13c.xsl:
  - Überladen – und damit Kontrolle - der „störenden“ Default-Regel

XSLT: Implizite Regeln

- Beobachtungen:
  - Die Ausgabe reduziert sich nun auf die Erzeugung der XML-Deklaration.
  - Der Textinhalt der Quelldatei ist nun verschwunden.
  - Offenbar wurde die implizite Schablonenregel außer Kraft gesetzt.
- Naheliegende Variante:
  - Können wir vielleicht die Attributwerte ausgeben?

XSLT: Schablonen

```
<?xml version="1.0"?>  <!-- FUNKTIONIERT NOCH NICHT -->
<xsl:stylesheet xmlns:xsl=
   "http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="text"/>
    <xsl:template match="text()"/>
    <xsl:template match="@*">
    Attribut: <xsl:value-of select="name(.)"/>
    Wert:     <xsl:value-of select="."/>
    Element:  <xsl:value-of select="name(..)"/>

    </xsl:template>
</xsl:stylesheet>
```

- ueb13d.xsl:
  - Ignorieren der Textknoten, Auflisten der Attribute + Kontext
  - zugleich ein Beispiel für Text-Erzeugung

XSLT: Schablonenregeln

- Beobachtungen:
  - Leiten Sie die Ausgabe in eine Datei ueb13d.txt
  - Die Textknoten bleiben abgeschaltet
  - Die Attributknoten werden offenbar so nicht gefunden! Dies könnte eine Störung sein …
  - Mit <value-of> wird zur Laufzeit ein konkreter Wert ausgegeben. Derartige Konstrukte erklären den Namen „Schablone“ (*template*).

XSLT: Kopieren von Teilbäumen

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl=
  "http://www.w3.org/1999/XSL/Transform"
  version="1.0">
<xsl:output method="xml"/>
   <xsl:template match="@* | node()">
     <xsl:copy>
       <xsl:apply-templates select="@*|node()"/>
     </xsl:copy>
   </xsl:template>
</xsl:stylesheet>
```

- ueb13e.xsl:
  - „Identische“ Kopie erzeugen (rekursiver Ansatz)

XSLT: Kopieren von Teilbäumen

- Auswertungen:
  - Leiten Sie die Ausgabe in Datei ueb13e.xml um.
  - Vergleichen Sie Quelle und Ziel, z.B. mittels
    - diff ueb13.xml ueb13e.xml
  - Unterschiede?
    - *root*-Element: *whitespace*-Normierungen, *encoding* (!) (nur beim Beispiel mit den *namespace*-Deklarationen)
    - Kommentar: Umlaut ü umcodiert, entsprechend *encoding*
    - Letztes *tag:* Zeilenende – *char data* außerhalb des Dokuments

XSLT: Kopieren von Teilbäumen

- Beobachtungen:
  - Alle wesentlichen Informationen wurden reproduziert
  - Die Unterschiede sind erwartete Folgen von Normierungen beim Wechsel Dokument – Datenmodell – Dokument.
  - Kontrolle über *encoding* des Zieldokuments?
    - Suchen Sie die Antwort selbst und testen Sie Ihr Ergebnis!
- Anmerkungen:
  - In der Praxis nutzt man <xsl:copy> eher zum Kopieren von Teilbäumen
  - Wieso sprechen wir hier von einem rekursiven Ansatz?
  - Mit <xsl:copy-of> gibt es eine nicht-rekursive, einfachere aber auch weniger flexible Alternative.

XSLT: Unser bisheriges „Vokabular“

- Zusammenstellung am Ende der Vorübungen:
  - <xsl:stylesheet>
  - <xsl:output method=…>
  - <xsl:template match=…>
  - <xsl:apply-templates select=…>
  - <xsl:value-of select=…>
  - <xsl:copy>, <xsl:copy-of>

Auswertungen

Auswertungsaufgaben
Anzeige-Aufgaben

XSLT: Auswertungen

- Vorbemerkungen
  - Die Auswertungsaufgaben (ueb13f) sind technisch anspruchsvoller als die Anzeigeaufgaben (ueb14).
  - Andererseits werden die Ergebnisse von ueb13f als Input für ueb14 benötigt.
  - Daher empfiehlt sich,
    - zunächst ueb13f nur nach Anleitung durchzuführen
    - dann ueb14 zu bearbeiten - und dabei sich die neuen XSLT-Sprachelemente anzueignen
    - schließlich den Code von ueb13f nachzuvollziehen und die (nunmehr geringeren) Neuerungen von XSLT zu erlernen.

XSLT: Auswertungsaufgaben

XSLT: Auswertungsaufgaben

- Erläuterungen:
  - Nach Abschluß der Vorübungen ueb13a-e soll nun mit ueb13f eine erste nützliche Aufgabe gelöst werden.
  - Ausgangspunkt ist wieder die Klausurauswertung.
  - Wie der Dateiname schon andeutet, ist ueb13f.xsl eine – wenn auch erhebliche – Weiterentwicklung von ueb13e.xsl.
  - Die Auswertung besteht aus mehreren Maßnahmen zur Vervollständigung und Plausibilitätskontrolle der Quelldaten.
  - Das Ergebnis – eine neue XML-Datei ueb13out.xml – wird als Input für die nächste Aufgabe dienen:
- Ausblick:
  - „Veröffentlichung“ des Klausurergebnisses im Internet, d.h. Erzeugung verschiedener s*tylesheets* zur Generierung von HTML-Output.

XSLT: Auswertungsaufgaben

- Aufgabe 13f:
  - Vervollständigung der Ergebnisliste:
    - 1) Punkte einer Aufgabe mit Teilaufgaben:
      - Element nachtragen,
      - dazu Summe berechnen
    - 2) Ergebnis: Attribut „Punkte“ berechnen/einfügen
  - Vervollständigung der Teilnehmerliste:
    - 3) Attribut „Punkte“ ermitteln/einfügen.
    - Studentenübung: Analog Attribut „Note“ ermitteln/einfügen.
    - Optional: Attribut „abgegeben“ auf „ja“ setzen, falls Punkte > 0.
  - Plausi-Checks / Fehler-Reports:
    - Fall: MaxPunkte von Aufgabe <> Summe MaxPunkte ihrer Teilaufgaben
    - Fall: Ergebnis mit MatrNr, die nicht in Teilnehmerliste erscheint
    - Fall: Ergebnis mit Wert <> Summe Werte der Aufgabenteile

XSLT: Auswertungsaufgaben

- „Geführte“ Übung 13f im Rahmen der Vorlesung
  - Interaktive Anleitungen zum Nachvollziehen
  - Kommentierte Code-Abschnitte zum Nachlesen
  - Diskussion der neuen XSLT-Elemente!
  - Bem.:
    - Als eigenständige Übung erst im kommenden Kurs vorgesehen
    - Eigener Studenten-Übungsteil hier: Erweiterung der Betriebsart '3' - Einfügen des Attributs „Note“ in Element „Klausurinfo“. Hinweis: Dies ist i.w. eine XPath-Übung.

XSLT: Auswertungsaufgabe

Hinweise und Kommentare zur Übung 13f:

- Dreistufiges Vorgehen
  - Abgebildet als drei Betriebsarten desselben *stylesheets*
  - Implementiert mit Parameter „`ctrl`“ = 1, 2 oder 3.
  - Verwenden Sie temporäre Dateien für die Zwischenschritte.
- Zum Nachvollziehen:

```
Xalan -p ctrl 1 ueb13.xml ueb13f.xsl > tmp1.xml
Xalan -p ctrl 2 tmp1.xml ueb13f.xsl > tmp2.xml
Xalan -p ctrl 3 tmp2.xml ueb13f.xsl > ueb13out.xml
```

- Zum Vergleichen / Erkennung der Wirkungen:

```
diff ueb13.xml tmp1.xml    # Zwei neue Zeilen
diff tmp1.xml tmp2.xml     # + neue Attr. in „Ergebnisse“
diff tmp2.xml ueb13out.xml # … und in „Klausurinfo“
```

XSLT: Neues „Vokabular“

- Hinzugekommen sind nun:
  <xsl:param name=… >,
  <xsl:variable name=… select=… >
  <xsl:choose>
  <xsl:when test=…>
  <xsl:otherwise>
  <xsl:element name=…>
  <xsl:attribute name=…>
  <xsl:apply-templates select=…>

XSLT: Anzeige-Aufgaben

XSLT: Anzeige-Aufgaben

- Aufgaben:
  - Einfache Ergebnistabelle
    - Spalten: Matr.Nr., Aufgabe (Block), Summe
    - Zeilen: Überschrift, Unter-Überschrift (Aufgabennr), Ergebnisse
    - Studentenübung: (Sortierübung) Zeilen nach Matr.Nr, bzw. nach Punktsumme sortiert ausgeben.
  - Komplexe Ergebnistabelle
    - KursID, KursName, Semester als Titelzeile ausgeben
    - Neue erste Spalte: Laufende Nr., von XSLT berechnen lassen
    - Neue Spalten im Block: Teilaufgaben
    - Studentenübung: Neue letzte Spalte ergänzen: Note
  - Klausurspiegel
    - Kleine Tabelle (2 x (5+1)), Noten+Teilnehmer vs. Anzahl
  - Teilnehmerliste („nicht öffentlich“)
    - Tabelle: Name, angemeldet, erschienen, abgegeben, Punkte

XSLT: Anzeige-Aufgaben

- „Geführte“ Übung 14
  - Interaktive Anleitungen
  - Kommentierte Code-Abschnitte
  - Als eigenständige Übung erst im kommenden Kurs vorgesehen; noch auszuarbeiten
- Eigenständige Übungsteile, vorläufig:
  - A) Ändern Sie die Sortier-Reihenfolge
  - B) Erweitern Sie die Ausgabe um eine Spalte für die Note
  - C) Erstellen Sie eine Teilnehmerliste (in HTML), indem Sie das Beispiel zur Ergebnisliste geeignet modifizieren.

XSLT: Anzeige-Aufgaben

- ueb14text.xsl
  - Textmodus-Übung, keine HTML-Kenntnisse nötig
  - Ziele
    - Einbettungen von <apply-templates> verdeutlichen die Wechselbeziehungen zwischen den Regeln
    - <text> zur geordneten, XML-konformen Textausgabe
    - Einige XPath-Ausdrücke in Aktion
    - Demo für die Erzeugung von neuer, nicht direkt in der Quelle vorhandener Information: Beispiel „Summenzeile“
  - Neu:
    - <apply-templates> in Aktion
    - <xsl:text>

XSLT: Anzeige-Aufgaben

- ueb14html1.xsl
  - Einfache Übung zur HTML-Generierung
    - Schreiben Sie den Output in Datei out14-1.html.
    - Betrachten Sie das HTML.Ergebnis mit dem Browser.
  - Bemerkungen
    - Inhaltlich analog zu ueb14text.xsl, nun mit den HTML-Möglichkeiten „verschönert“
    - Typisches Beispiel für eine Tabellenerzeugung
    - Einbettung gängiger HTML-Elemente und –Attribute in den Text des XML *stylesheet*: Saubere Trennung über Präfix / Namensraum!
  - Neu:
    - Einige HTML-Elemente / -Attribute: <html>,<title>,<body>, <h1>, <p>,
; <table>, <th>, <tr>, <td>

XSLT: Anzeige-Aufgaben

- ueb14html0.xsl
  - Noch einfachere Übung zur HTML-Generierung
    - Schreiben Sie den Output in Datei out14-0.html.
  - Bemerkungen
    - Analog zu ueb14html1.xsl, nun drastisch vereinfacht durch Auslassen einiger XSLT-Elemente wie <xsl:stylesheet>.
    - Besonderheit im HTML-Umfeld, um den Umstieg nach XSLT zu erleichtern.
    - Schablonengedanke besonders gut erkennbar, auch wenn der Preis aus Einschränkungen besteht.
    - Verwandtschaft mit anderen in HTML eingebetteten Skriptsprachen wird so leichter erkennbar.

XSLT: Anzeige-Aufgaben

- ueb14html2.xsl
  - Komplexere Übung zur HTML-Generierung
    - Schreiben Sie den Output in Datei out14-2.html
  - Neu:
    - <xsl:sort>
    - <xsl:for-each>
    - „mode“-Attribut von <apply-templates> und <template>
    - <xsl:message>
    - Auswertung einer Variablen (als Attributwert)
    - Vorgriff auf Übung 15a (wird hier gebraucht)

XSLT: Anzeige-Aufgaben

- Ueb14-inc.xml
  - Impliziter *stylesheet*-Aufruf, analog CSS
  - Laden Sie diese xml-Datei direkt mit Ihrem Browser. Ergebnis:
    - Anzeige fast wie im Fall ueb14html2 via Xalan
    - Unterschiede in den Meldungen zum XSLT-Prozessor! → Server- vs. Client-Ansatz erkennbar unterschiedlich!
  - Neu:
    - <xml-stylesheet href="…" type="text/xsl">

XSLT: Anzeige-Aufgaben

Geplante Erweiterungen:

- Optionale Aufgabe:
  - Shakespeare-Beispiel nach HTML wandeln
  - Dazu ein vorhandenes *stylesheet*
    - verstehen lernen
    - erweitern und modifizieren
- Optionale Aufgabe:
  - Formatierung der XML-Spezifikationen
  - Bei Eignung als Grundlage für FO-Übung?

XSLT: Technischer Anhang

Sprachelemente und Funktionen
Erweiterungen
Einbinden externer Datenquellen

XSLT: Elemente und Funktionen

- Hinweise zur Verwendung des Materials
  - Die folgenden Aufstellungen sind keine Erklärungen. Sie verstehen sich als schnelle Hilfe zur Suche nach konkretem Material.
  - Verwenden Sie die XSLT-Spezifikationen oder falls vorhanden einschlägige Bücher zum Nachschlagen der Einzelheiten!
  - Elemente, die nicht in den Übungen behandelt wurden, sind farblich hervorgehoben.

XSLT-Elemente, gegliedert

- Definition und Verwendung von Schablonenregeln
  - <xsl:template>
  - <xsl:apply-templates>
  - <xsl:call-template>
  - <xsl:apply-imports>
- Elemente zur Strukturierung von stylesheets
  - <xsl:stylesheet>
  - <xsl:include>
  - <xsl:import>
- Ausgabeerzeugung
  - <xsl:value-of>
  - <xsl:element>
  - <xsl:attribute>
  - <xsl:attribute-set>
  - <xsl:comment>
  - <xsl:processing-instruction>
  - <xsl:text>
- Kopieren von Teilbäumen
  - <xsl:copy>
  - <xsl:copy-of>

XSLT-Elemente, gegliedert

- Umgang mit Variablen und Parametern
  - <xsl:variable>
  - <xsl:parameter>
  - <xsl:with-param>
- Sortieren und Nummerieren
  - <xsl:sort>
  - <xsl:number>
- Suchen und Finden
  - <xsl:key>
- Sonstiges
  - <xsl:message>
  - <xsl:namespace-alias>
- Bedingte Verarbeitung
  - <xsl:if>
  - <xsl:choose>
  - <xsl:when>
  - <xsl:otherwise>
  - <xsl:fallback>
  - <xsl:for-each>
- Outputsteuerung
  - <xsl:output>
  - <xsl:decimal-format>
  - <xsl:preserve-space>
  - <xsl:strip-space>
  - <xsl:transform>

XSLT-Funktionen

- Generell: Alle XPath *core functions*
  - Näheres siehe dort!
- Zusätzlich:
  - Allgemeine XSLT-Erweiterungen
  - Optionale, prozessorspezifische oder auch benutzerdefinierte Erweiterungen
- Bemerkungen zur folgenden Zusammenstellung:
  - Keine optionalen Funktionen
  - Kennzeichnung, ob neu (XSLT) oder behandelt (XPath).

XSLT-Funktionen, gegliedert

- Datentypkonvertierungen
  - boolean
  - format-number
  - number
  - string
- Arithmetische Funktionen
  - ceiling
  - floor
  - round
- Boolesche Funktionen
  - false
  - true
  - not
- Aggregierungen
  - sum
  - count
- Stringverarbeitung
  - concat
  - contains
  - normalize-space
  - starts-with
  - string-length
  - substring
  - substring-before
  - substring-after
  - translate
- Kontextliefernde Funktionen
  - current
  - last
  - position

XSLT-Funktionen, gegliedert

- Knotennamen und *identifier* erhalten
  - generate-id
  - lang
  - local-name
  - name
  - namespace-uri
  - unparsed-entity-uri
- Knoten suchen/liefern
  - document
  - key
  - id
- Informationen über den XSLT-Prozessor erhalten
  - element-available
  - function-available
  - system-property

XSLT-Erweiterungen

- Allgemein:
  - Sowohl XSLT–Funktionen als auch XSLT-Elemente können ergänzt werden.
  - Manche Prozessoren besitzen schon eingebaute Erweiterungen.
- Bei der Verwendung beachten:
  - Erweiterungen sind schlecht portabel!
  - Neue Elemente und Funktionen müssen mit separaten Namensräumen / Präfixwerten vom Standard unterschieden werden.
  - Verwenden Sie element-available() bzw. function-available(), um Verfügbarkeiten zur Laufzeit zu ermitteln.

XSLT-Erweiterungen

- Beispielcode (Fragment) für eine Elementerweiterungen des Prozessors saxon:
  - ```
    <xsl:template … >
      <saxon:while test="…"
             xmlns:saxon="http://icl.com/saxon">
             …
      </saxon:while>
    </xsl:template>
    ```
  - („while“, z.B. in Ergänzung zu „for-each“)
- Beispielcode (Fragment) für eine Funktionserweiterungen des Prozessors Xalan:
  - ```
    <xsl:template … >
      <xsl:for-each
             test="xalan:intersection(./@foo, ./@bar)"
             xmlns:xalan="http://xml.apache.org/xalan">
             …
      </xsl:for-each>
    </xsl:template>
    ```
  - („intersection“ liefert die Schnittmenge)

XSLT-Erweiterungen

- Informationen über die Laufzeitumgebung
  - Verwenden Sie system-property(), um Näheres über den XSLT-Prozessor selbst herauszufinden.
- Argumente von system-property():
  - xsl:version
    - Zahl mit der XSLT-Version
  - xsl:vendor, xsl:vendor-url:
    - Strings mit dem Herstellernamen des XSLT-Prozessors bzw. seiner WWW-Adresse
  - (weitere)
    - Implementierungsabhängig
    - Ursprünglich war vorgesehen, so Informationen über das Betriebssystem zugänglich zu machen (daher der Funktionsname).
    - Einige Hersteller könnten derartige Erweiterungen anbieten, verlassen sollte man sich nicht darauf.
- Übung:
  - Schreiben Sie ein kleines *stylesheet* ueb15a.xsl, das die drei Werte für Ihren XSLT-Prozessor ausgibt. Verwenden Sie ueb13b.xsl als Basis.

XSLT: Variablen und Parameter

- Variablen bzw. Parameter werden
  - mit <xsl:variable> bzw. <xsl:param> angelegt. Beispiel:

```
<xsl:variable name="foo"
      select=" 'eine Zeichenkette' "/>
```

    (man beachte die doppelte Quotierung).
  - mit einem vorangestellten $ in XPath-Ausdrücken referenziert. Beispiele:

```
<xsl:text>Der Wert von foo ist:
      <xsl:value-of select="$foo"/> </xsl:text>
<img src={$basedir}/{href} width={size/@width}/>
```

  - Man beachte die geschweiften Klammern {}: Sie ermöglichen die Auswertung von Ausdrücken auch bei Attributwert-Angaben! Auch Referenzen auf Variablen sind schließlich Ausdrücke…

XSLT: Variablen und Parameter

- Variablen können nicht aktualisiert werden!
  - "Funktionales Programmieren“ à la Lisp, Transformation: Output = „Funktion“ des Inputs, O = S(I)
  - Eigentlich „lokale Konstanten“
  - Konsequenz: Rekursionen statt Iterationen verwenden!

XSLT: Variablen und Parameter

- Unterscheide globale und lokale Variablen!
  - Globale Variablen müssen auf *top-level* angelegt werden, d.h. mit <xsl:variable> als Kindelement von <xsl:stylesheet>
  - Variablen, die innerhalb von <template>-Elementen angelegt werden, sind lokal:
    - Sie sind wirksam für den Kontextknoten und alle „*following siblings*“ sowie deren „Nachkommen“
    - Sie sind nicht wirksam für die Nachkommen des Kontextknotens selbst sowie für seine „*preceding siblings*“.
    - Sie verlieren ihren Bezug mit dem Ende-*tag* des Elternknotens des Kontextknotens.
    - Sie werden ebenfalls ungültig (*out of scope*) außerhalb ihres XSLT-Elternelements.

XSLT: Variablen und Parameter

- Besonderheiten von Parametern
  - Parameter unterscheiden sich nur in ihrem Initialisierungsverhalten von Variablen
  - Während Variablem einmal mit einem festen Wert belegt werden, kennen Parameter eine Default-Initialisierung, die „von außen“ überschrieben werden kann.
- Default-Initialisierung und Überschreiben
  - Die Default-Initialisierung entspricht der Belegung per „select“ im <xsl:param>-Element, analog zu Variablen.
  - Überschreiben globaler Parameter
    - Implementierungsabhängig, z.B. bei Xalan per Kommandozeilenoption –p *par-name par-value*
  - Überschreiben lokaler Parameter
    - Durch <xsl:with-param> als Kind-Element von <xsl:apply-templates> oder <xsl:call-template>

XSLT: Einbinden externer Daten

- Die Funktion document()
  - … kann mit verschiedenen Argumenten aufgerufen werden. Typisch: URI
  - … bewirkt ein Parsen des übergebenen XML-Dokuments, die Bildung eines Datenmodells, und die Rückgabe der spezifizierten Knotenmenge, z.B. des *root*-Knotens.
  - … ermöglicht somit die Einbindung von Daten außerhalb des aktuellen Dokuments.
  - … birgt enorme Möglichkeiten, z.B. durch
    - Verkettung von document()-Aufrufen
    - Parametrisierung der URI, etwa durch User-Interaktion
    - dynamische Erzeugung zu ladender Daten, etwa indem der URI auf ein CGI-Skript oder ein Java Servlet zeigt und Parameter codiert – Datenbankanbindungen sind so möglich!.

XSLT: Einbinden externer Daten

- Codebeispiel zu document() aus Michael Kay's Buch:

```
<book>
  <review date="1999-12-28" publication="New York
   Times" text="reviews/NYT/19991228/rev3.xml"/>
  <review date="2000-01-06" publication="Washington
   Post" text="reviews/WPost/20000106/rev12.xml"/>
</book>

<xsl:template match="book">
  <xsl:for-each select="review">
    <h2>Review in<xsl:value-of select="@publication"/>
    </h2>
    <xsl:apply-templates select="document(@text)"/>
  </xsl:for-each>
</xsl:template>
```

XSLT: Einbinden externer Daten

- Wirkung:
  - Das *template* für „book“ erzeugt eine Folge von „Reviews“:
    - Zunächst Titel (h2) mit Quellenangabe
    - Dann Ausgabe des referenzierten XML-Dokuments (!)
- Bemerkungen:
  - Damit die Ausgabe funktioniert, müssen die referenzierten Dokumente strukturell zu den Schablonenregeln des aktuellen *stylesheet* passen.
  - Im einfachsten Fall fügt man schlicht fehlende Regeln hinzu.
  - Möglichkeiten zur Lösung von evtl. Namenskollisionen:
    - Verschiedene Namensräume verwenden
    - Verwendung von „mode“ zur Unterscheidung von Regeln, etwa:

```
<xsl:apply-templates select="document(@text)"
  mode="review"/>
```