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.

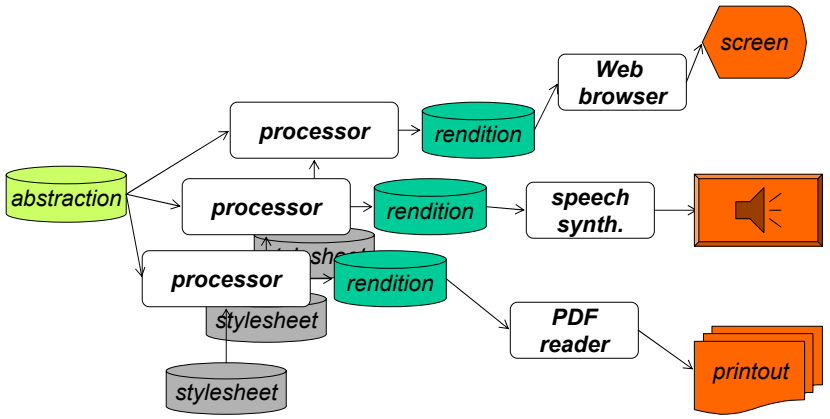
    Diese Transformation von XML-Dokumenten erwies sich als eigenständige Aufgabe, die auch unabhängig von Formatierungen ihren Wert besitzt, und wurde daher als eigene Spezifikation formuliert.
  - **Formatting Objects (FO)**

    Die Spezifikation zu FO bildet den eigentlichen Kern von XSL.

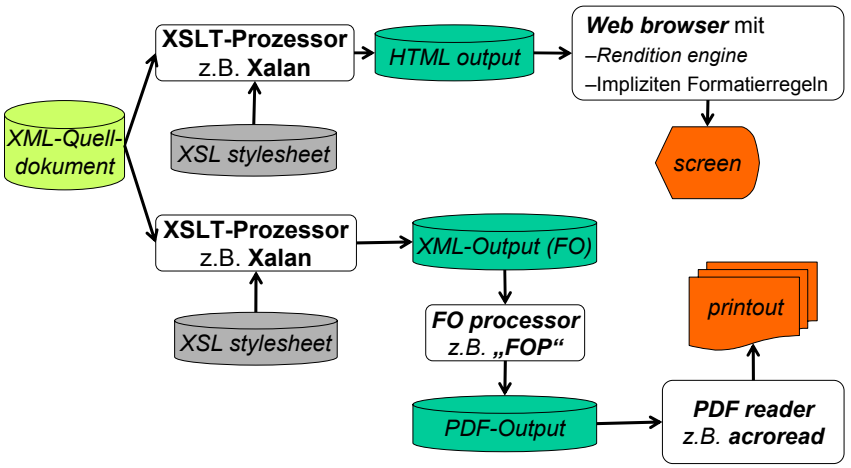
    Hier wird die (XML-basierte) Beschreibungssprache für die Präsentation von Daten definiert.

    XSL-FO ist komplex und umfangreich. Zu Verständnis ist Hintergrundwissen zu allgemeinen Darstellungsfragen erforderlich

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

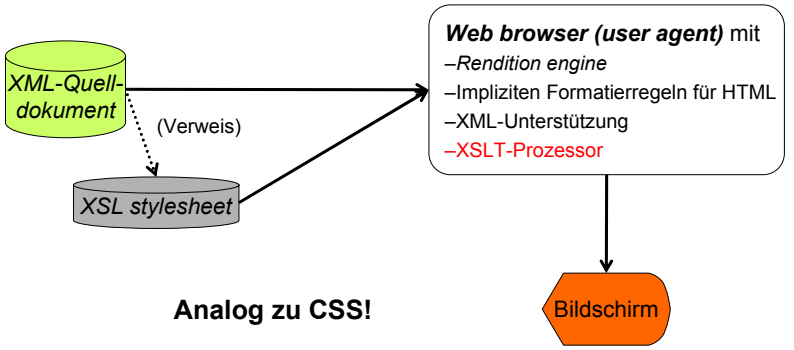


… und nun konkreter:



XSL

… oder auch:

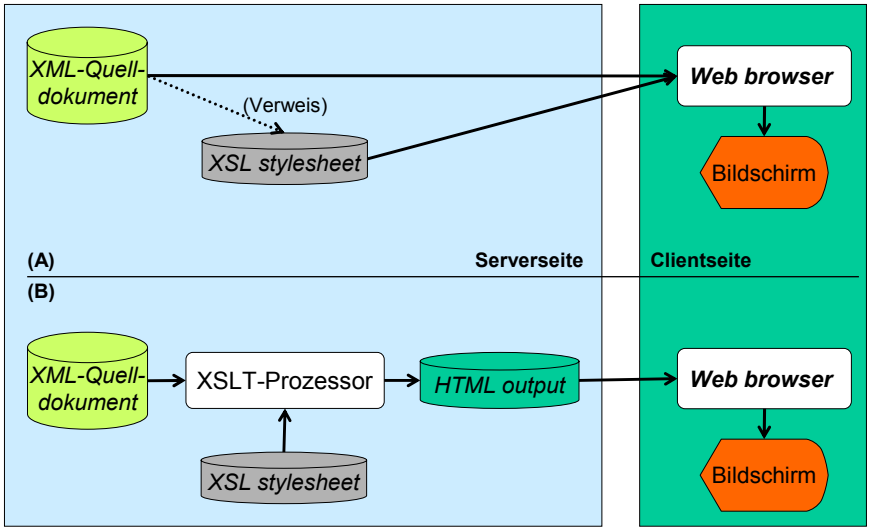


Analog zu CSS!

XSLT

- Demo
  - „Othello“-Szene
    („Wrox“-Buchbeispiel aus Kapitel 9)
  - „Tempest“-Werk, mit gleichem *stylesheet*
  - mit Firefox und/oder „xsltproc“ 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 Server-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
  - 2005-11-03: Version 2.0 (nun CR)
    Verwendet XPath 2.0 und unterstützt XML Schema
    Autor: Michael Kay, Saxonica Ltd.

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 2nd Edition“ von Michael Kay, Wrox Press, 2002.
  - 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, Einschränkungen, usw. in den Spezifikationen!

XSLT: Vorbereitung

- Vorlesungsübung
  - Legen Sie ein Unterverzeichnis „10“ an
  - Kopieren Sie „10-tempest.xml“ und „10-shaksper.dtd“ vom entsprechenden Verzeichnis des Dozenten dorthin.

    Es handelt sich um leicht erweiterte Varianten von Übung 03.
  - Legen Sie im folgenden die Stylesheet-Dateien unter dem Namen „10-*x*.xsl“ an, mit *x*=(a, b, c, …)
  - Verwenden Sie folgende Aufrufe des XSLT-Prozessors:

```
xalan -in src.xml -xsl sheet.xsl # Ausgabe nach stdout
xalan -in src.xml -xsl 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.

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 10-empty.xsl an.
- Vergleichen Sie die Ausgabe von Xalan für
  - method="xml",
  - method="html" und
  - method="text" sowie zur Prüfung des XSLT-Prozessors noch:
  - 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 – auch ohne Regeln!**
    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 Sammlung von **Schablonenregeln** (*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!
  - „Variablen“ lassen sich nicht mehr ändern → Stack!
  - 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** von Element „template“
    - Die Attributwerte sind i.w. XPath-Ausdrücke
  - **Schablone** (*template*)
    - Die eigentliche Anweisung, was mit den gefundenen Knoten geschehen soll.
    - Abgebildet schlicht als Elementinhalt von „template“.
  - 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>Hallo</h1>
    <p>Hallo Welt!
    <br/>Test zu "br": Zweite Zeile…</p>
    </body></html>
  </xsl:template>
</xsl:stylesheet>
```

- 10-hello-h.xsl:
  - Eine einfache Schablonenregel und ein Beispiel für HTML-Erzeugung

XSLT: Schablonenregeln

- Beobachtungen:
  - Leiten Sie die Ausgabe in eine Datei 10-hello.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 „br“ 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!**
  - Sie sind die Ursache des Xalan-Outputs bei Beispiel „empty“ trotz Fehlens jeglicher Schablonenregeln!
- Priorisierung:
  - Analog zu „importierten“ Regeln

    Hinweis: <xsl:import>
  - **Interne Regeln haben Vorrang!**
- Konsequenz:
  - **Überladen der eingebauten Regeln deaktiviert die impliziten Regeln.**

XSLT: Implizite Regeln

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

- Selektiert den root-Knoten und alle Elementknoten.
- **<xsl:apply-templates/>** ruft Schablonenregeln auf

  Wenn nicht weiter eingeschränkt, für alle selektierten Knoten.

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

- Analog, für jede Einschränkung mittels `mode`-Attribut (vgl. Kap. 5.7)
- Bem.: „mode“ gestattet die Sonderbehandlung bestimmter Elemente unter ausgewählten Bedingungen.

XSLT: Implizite Regeln

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

- Selektiert alle Text- und Attributknoten des Kontextknotens
- **<xsl:value-of>** gibt den Stringwert (hier: des Kontextknotens „.“) aus!
- Diese Regel verursachte unsere Outputs!

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

- Selektiert alle PI- und Kommentarknoten.
- „*Empty element*“, ohne Schablone → 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>
```

- 10-empty-2.xsl:
  - Überladen – und damit Kontrolle - der „störenden“ Default-Regel

XSLT: Implizite Regeln

- Beobachtungen:
  - Die Ausgabe reduziert sich nun auf die Erzeugung der XML-Deklaration (nur im Fall „xml“-Modus).
  - 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"?>
<xsl:stylesheet xmlns:xsl=
  "http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="text"/>
   <xsl:template match="*|/">
     <xsl:apply-templates select="*|@*"/>
   </xsl:template>
   <xsl:template match="text()"/>
   <!-- So sieht man mehr als mit der Defaultregel: -->
   <xsl:template match="@*">
     Attribut: <xsl:value-of select="name(.)"/>
     Wert:     <xsl:value-of select="."/>
     Element:  <xsl:value-of select="name(..)"/>
   </xsl:template>
</xsl:stylesheet>
```

- 10-empty-3.xsl:
  - Ignorieren der Textknoten, Auflisten der Attribute mitsamt Kontext
  - zugleich ein Beispiel für Text-Erzeugung

XSLT: Schablonenregeln

- Beobachtungen:
  - Leiten Sie die Ausgabe in eine Datei 10-empty-3.txt
  - Die Textknoten bleiben abgeschaltet
  - 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>
```

- 10-copy-1.xsl:
  - „Identische“ Kopie erzeugen (rekursiver Ansatz)

XSLT: Kopieren von Teilbäumen

- Auswertungen:
  - Leiten Sie die Ausgabe in Datei 10-copy-1.xml um.
  - Vergleichen Sie Quelle und Ziel, z.B. mittels
    diff 10-tempest.xml 10-copy-1.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
- Bemerkungen:
  - Eigentlich keine vollständige Kopie: Namespace-Knoten würden fehlen, *doctype*-Deklaration fehlt (Parser!)
  - <copy> ermöglicht eine kontrollierte, selektive Kopie!

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 Einführungsbeispiele:

  <xsl:stylesheet>

  <xsl:output method=…>

  <xsl:template match=…>

  <xsl:apply-templates select=…>

  <xsl:value-of select=…>

  <xsl:text>

  <xsl:copy>, <xsl:copy-of>

Transformation in HTML mit XSLT

Das Shakespeare-Beispiel
in einfachen Teilschritten

Tafelbild

- Planung:

PLAY/TITLE	→	h1, zentriert
ACT/TITLE	→	h2, zentriert, hline vorher
SCENE/TITLE	→	h3
SPEECH	→	p
SPEAKER:	→	bold, mit Doppelpunkt
LINE:	→	br (neue Zeile)
STAGEDIR	→	p, em (kursiv)

Rest: erst mal ignorieren…

Tafelbild

- Verfeinerung:

PLAY/TITLE →	h1, zentriert
ACT/TITLE →	h2, zentriert, hline vorher
PERSONAE/TITLE:	wie ACT/TITLE , dann …
PERSONAE/PERSONA →	wie LINE
PGROUP →	neue Zeile auslösen
PGROUP/PERSONA →	„inline“-artig, plus Komma bzw. Bindestrich (bei letzter Instanz)
GRPDESCR →	Wert kursiv
SCNDESCR →	Wie SCENE/TITLE

XSLT: Unser bisheriges „Vokabular“

- Neue Sprachelemente, Erweiterungen:
  - <xsl:choose>,
  - <xsl:when test=…>,
  - <xsl:otherwise>
  - <xsl:if test=…> <!– Analog choose/when -->
  - <xsl:template name=…>
  - <xsl:call-template name=…>

XSLT: Ausgewählte Kapitel

Variablen und Parameter
Sprachelemente und Funktionen
Erweiterungen
Einbinden externer Datenquellen

- **Variablen und Parameter**
  - werden mit <xsl:variable> bzw. <xsl:param> angelegt.
    Beispiele:

```
<xsl:variable name="foo"
      select="'eine Zeichenkette'"/>
```
(man beachte die doppelte Quotierung).

```
<xsl:param name="bar"/>
```

  - werden mit einem vorangestellten **$** in XPath-Ausdrücken referenziert. Beispiele:

```
<xsl:text>Der Wert von foo ist: </xsl:text>
<xsl:value-of select="$foo"/>
```

XSLT: Variablen und Parameter

- **Attributwert-Schablonen**
  - In der Konstruktion
    ```
    <img src="{$basedir}/{href}" width="{size/@width}"/>
    ```
    beachte man die geschweiften Klammern {}:

    Diese Attributwert-Schablonen erlauben die Auswertung von XPath-Ausdrücken auch in Attributwert-Angaben!

  - Auch Referenzen auf Variablen sind XPath-Ausdrücke!

- Besonderheiten:
  - Kein „*nesting*“, also keine XPath-Ausdrücke in {…}, die selbst {…} enthalten!
  - *Escaping*: {{ oder }} innerhalb von Attributwert-Schablonen ergibt wörtlich { bzw. } als Ausgabe.

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.
  - „**Scope**“ lokaler Variablen:
    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

- Wichtig:
  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!
  - Vorteile: Keine Seiteneffekte, gut optimierbar.

XSLT: Variablen und Parameter

- Besonderheiten von Parametern
  - **Parameter unterscheiden sich nur in ihrem Initialisierungsverhalten von Variablen**
  - Während Variablen 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: Variablen und Parameter

- Rekursiver Programmierstil und die Verwendung lokaler Parameter – Ein Beispiel:

  Aus.: M. Kay, XSLT, Kap. 4, Beispiel „longest-speech.xsl“:

Aufgabenstellung:

Gesucht ist die größte Anzahl aufeinander folgender Textzeilen aller Akteure in einem gegebenen Shakespeare-Drama.

Ansatz:

Kindelemente „LINE“ der Elternelemente „SPEECH“ zählen, Maximum dieser Werte ausgeben,

Rekursion zur Ermittlung des Maximums

Online-Demo mit xsltproc + Diskussion des Codes:

Eine Lösung der in XPath vermissten Funktion max()!

Code von longest-speech.xsl zum Nachlesen

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

 <xsl:template name="max">
  <xsl:param name="list"/>
  <xsl:choose>
   <xsl:when test="$list">
    <xsl:variable name="first"
        select="count($list[1]/LINE)"/>
    <xsl:variable name="max-of-rest">
     <xsl:call-template name="max">
      <xsl:with-param name="list"
          select="$list[position()!=1]"/>
     </xsl:call-template>
    </xsl:variable>
    <xsl:choose>
     <xsl:when
       test="$first &gt; $max-of-rest">
      <xsl:value-of select="$first"/>
     </xsl:when>
     <xsl:otherwise>
      <xsl:value-of select=
        "$max-of-rest"/>
     </xsl:otherwise>
    </xsl:choose>
   </xsl:when>
   <xsl:otherwise>
    0
   </xsl:otherwise>
  </xsl:choose>
 </xsl:template>

 <xsl:template match="/">
  <xsl:text>Longest speech
   is <xsl:text/>
  <xsl:call-template name="max">
   <xsl:with-param name="list"
        select="//SPEECH"/>
  </xsl:call-template>
  <xsl:text/> lines.
 </xsl:template>

</xsl:transform>
```

Kern der Rekursion

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>` (Synonym zu „stylesheet“)

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` (s.u.!)
  - `key`
  - `id`
- Informationen über den XSLT-Prozessor erhalten
  - `element-available`
  - `function-available`
  - `system-property`
  - – (Vgl. Praktikum!)

XSLT-Erweiterungen

- Allgemein:
  - – XSLT-Prozessoren können sowohl XSLT–Funktionen als auch XSLT-Elemente ergänzen.
  - – 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“: in Ergänzung zu „for-each“ oder statt „choose/when/otherwise“
- 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.

XSLT-Erweiterungen

- Verwendung von system-property() – ein Beispiel:

```
<xsl:template match="/">
   <xsl:value-of select=
    "system-property('xsl:vendor')"/>
  </xsl:template>
```

- Hinweis:
  - Siehe Übung 11, Teil A.
  - Demo zu Übung 11A:
    Kommandozeile (CygWin bash, xsltproc)
    Web Browser (IE 6, Firefox)

XSLT-Erweiterungen

- EXSLT: Eine pragm. Initiative zur Erweiterung von XSLT 1.x
  - Quelle: http://www.exslt.org
  - Module:

dates and times:	28 Funktionen
dynamic:	6 Funktionen
common:	3 Funktionen
functions:	3 Funktionen
math:	18 Funktionen, incl. min() und max()
random:	1 Funktion
regular expressions:	3 Funktionen
sets:	6 Funktionen
strings:	8 Funktionen

  - Besuch der Website, sofern die Zeit es gestattet
  - Vorteil: Vermeidung zahlreicher proprietärer Erweiterungen
  - Ausblick: Erfahrungen von EXLT gehen ein in XSLT 2.0

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 (Bsp.: "Photoalben")

    Parametrisierung des 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()`** von Michael Kay:

```
<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"/>
  <!-- usw. -->
</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"/>
```


XSL-FO

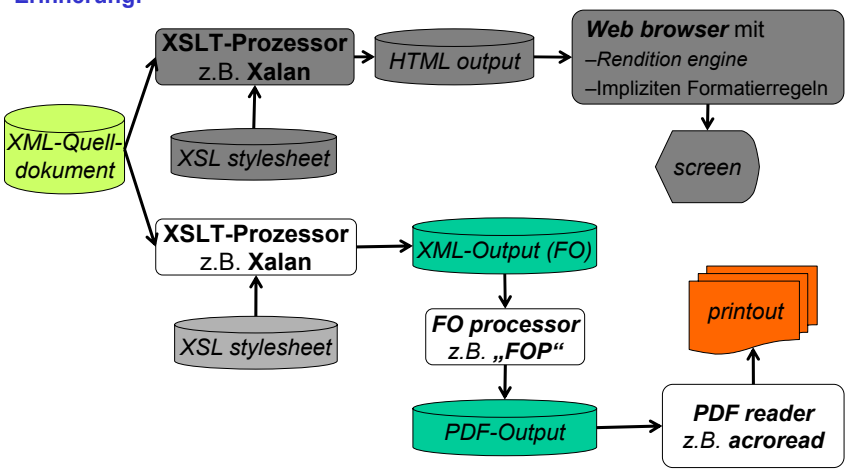
Formatting Objects

XSL-FO

- XSL-FO wird im Rahmen dieses Kurses nicht näher besprochen. Die Gründe sind:
  - Es handelt sich um ein Spezialthema, das erhebliches Fachwissen zur Seitengestaltung voraussetzt, um akzeptable Ergebnisse zu erzielen.
  - Vorhandene Lösungen und Produkte sind offenbar noch nicht ausgereift. Das Thema ist noch zu jung für die allgemeine Informatik-Grundausbildung.
- Wir beschränken uns auf eine reine Anwenderrolle
  - Man sollte vielleicht (noch) nicht lernen, wie FO im Einzelnen funktioniert, wohl aber, wie man damit umgeht.
  - Das allgemeine Vorgehen schauen wir uns anhand zweier Demo-Beispiele an.

XSL-FO: Ablaufschema

Erinnerung:



XSL-FO: Demo

- Ein Beispiel aus der Vertiefungs-LV „WBA“
  - DTD
    DocBook V 4.x
  - Stylesheets (sowohl für HTML als auch FO)
    DocBook V 4.x
  - XML-Editor:
    Beliebig, z.B. Emacs
  - XSLT-Prozessor:
    Xsltproc oder xalan
  - FO-Prozessor:
    FOP (Java-Anwendung) der Apache Foundation
    Erzeugt insb. Adobe PDF-Format
  - PDF-Viewer
    Acrobat Reader

XSL-FO: Demo

- Demo 1:
  - Wandlung eines komplexen XML-Dokuments (c‘t-Artikel, gesetzt in DocBook XML) mittels XSL *stylesheet* (XSLT) in eine Serie von HTML-Dateien.
  - Anzeige des Ergebnisses per Browser
- Demo 2:
  - Wandlung eines komplexen XML-Dokuments (c‘t-Artikel, gesetzt in DocBook XML) mittels XSL stylesheet (XSLT) in eine XSL-FO Datei.
  - Konvertierung der FO-Datei in eine PDF-Datei
  - Kurzer Eindruck von der FO-Datei
  - Anzeige des Ergebnisses mit PDF-Viewer