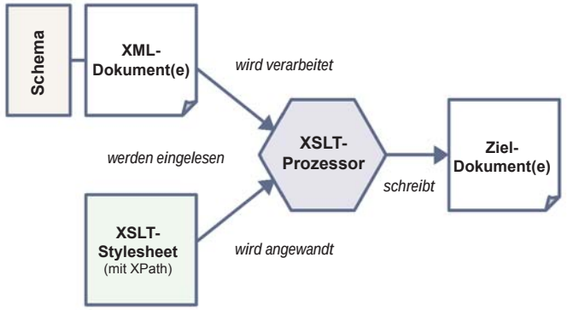
X-Technologien – Überblick



Mit Hilfe eines Schemas (z.B. DTD, XML Schema) können Typen von XML-Dokumenten (z.B. TEI, KML) definiert werden. Durch die Verknüpfung mit dem Schema wird das XML-Dokument diesem Dokumenttypen zugeordnet. Seine Struktur und Teile des Inhalts sind im Schema durch Regeln festgelegt und werden von einem Parser bei der Verarbeitung kontrolliert.

Einzelne oder mehrere XML-Dokumente können mit XSLT (unter Verwendung von XPath) transformiert werden. Das Ergebnis der Umwandlung sind Zieldokumente in einem textbasierten Format: Textdateien, XML-Dokumente, HTML-Dokumente, etc. So ist eine Trennung von XML-Dokumenten (die abstrakes Wissen enthalten) und bestimmten Ausgabeformaten bzw. Präsentationsformen möglich.

Für die Transformation werden das XML-Dokument und das XSLT-Dokument eingelesen („geparst“). Die XML-Dokumente werden verarbeitet, indem die XSLT-Anweisungen (und XPath-Ausdrücke) von einem XSLT-Prozessor auf den XML-Baum angewandt werden. Das Ergebnis der Transformation wird als Zieldokument geschrieben („serialisiert“).

2015
Zweite, überarbeitete und erweiterte Ausgabe
Autoren: Ulrike Henny, Patrick Sahle
Gestaltung: Markus Schnöpf
Satz: Ulrike Henny

Institut für Dokumentologie und Editorik e.V.
c/o Cologne Center for eHumanities (CCeH)
Universität zu Köln
Universitätsstraße 22
50923 Köln

XSLT – Extensible Stylesheet Language Transformation

XSLT ist eine Transformationssprache für XML-Dokumente. Ein XSLT-Prozessor verarbeitet ein XML-Dokument gemäß der Angaben in einem XSLT-Dokument und erzeugt ein Ergebnisdokument (oder mehrere) in einem Format (z.B. Plain Text, XML, HTML), welches vom XSLT-Dokument bestimmt wird.

Beispiel

```
❶ <?xml version="1.0" encoding="UTF-8"?>
❷ <xsl:stylesheet xmlns:xsl="http://www.
  w3.org/1999/XSL/Transform" version="2.0">
❸  <xsl:template match="/">
    ❹ Vorkommende Elementnamen:<br/>
      <ul>
     ❺ <xsl:for-each select="//*">
       ❻ <li>
           <xsl:value-of select="name()"/>
         </li>
       </xsl:for-each>
      </ul>
    ❼ <xsl:apply-templates/>
  </xsl:template>
❽  <xsl:template match="titel">
❾   <h1>Titel: <xsl:value-of select="."/></h1>
  </xsl:template>
</xsl:stylesheet>
```

❶ Ein XSLT-Stylesheet ist ein XML-Dokument!

❷ Wurzelelement `xsl:stylesheet`, hier: XSLT 2.0

❸ Eine „Schablone“ (`xsl:template`) trifft auf die Wurzel (`"/"`) des XML-Dokuments zu

❹ In das Ergebnisdokument wird etwas geschrieben (hier: ein String, ein HTML-Zeilenumbruch, das öffnende Tag für eine HTML-Liste)

❺ Für jedes Element in einem Knotensatz (hier: `"//*"` = beliebige Elemente in beliebiger Tiefe) soll das getan werden, was innerhalb der `xsl:for-each`-Schleife steht

❻ Es wird etwas (hier: HTML-Listeneinträge) in das Ergebnisdokument geschrieben. Darin wird ein Wert ausgegeben (hier: der Name des jeweils aktuellen Elements)

❼ Berücksichtige ggf. weitere, in diesem Kontext passende Schablonen

❽ Eine Schablone, die auf das Element `titel` anzuwenden ist

❾ Es wird etwas (hier: eine HTML-Überschrift) in das Ergebnisdokument geschrieben. Darin wird ein Wert ausgegeben (hier: der Text-Inhalt des aktuellen Knotens)

→ Dieses Beispiel-XSLT produziert eine Liste mit Elementnamen und gibt Überschriften aus

Wichtige XSLT-Elemente

`<xsl:template match="XPath-Ausdruck" name="string">`
– eine Schablone, ggf. mit Namen; darin ggf. Parameterannahme mit `<xsl:param name="string"/>`

`<xsl:apply-templates select="XPath"/>`
– berücksichtige in diesem Kontext weitere Schablonen (ggf. für bestimmte Elemente, die in `select` angegeben werden)

`<xsl:value-of select="XPath"/>`
– hole etwas aus dem XML-Dokument und schreibe es in das Ergebnisdokument. Kurzsyntax innerhalb von Attributkonstruktionen: *attributname*`="{XPath}"`

`<xsl:for-each select="XPath">`
– tue etwas für jedes Element einer Knotenmenge

`<xsl:sort select="XPath" data-type="text|number" order="ascending|descending"/>`
– sortiert eine Knotenmenge. Leeres Element! Unmittelbar nach `xsl:for-each` oder `xsl:apply-templates`

`<xsl:if test="XPath">`
– führe die folgenden Anweisungen nur aus, wenn die Bedingung in `test` erfüllt ist

`<xsl:choose>`
– wählt unter verschiedenen Bedingungen, die von den Kindelementen `<xsl:when test="XPath">` (mehrfach möglich) und `<xsl:otherwise>` beschrieben werden

`<xsl:variable name="string">`
– eine Variable; der Variableninhalt kann innerhalb des Elements aufgebaut werden oder im Attribut `name="XPath"`

`<xsl:element name="string">`
– konstruiert ein Element im Ergebnisdokument

`<xsl:attribute name="string">`
– konstruiert ein Attribut im Ergebnisdokument

`<xsl:text>`
– schreibt Zeichendaten in das Ergebnisdokument

`<xsl:output method="xml|html|text|..."/>`
– gibt an, wie das Ergebnisdokument ausgegeben werden soll (mit verschiedenen weiteren Attributen)

Häufige Fehlerquellen

- Der aktuelle Kontextknoten bei der Verarbeitung ist ein anderer, als man vermutet (keine Ausgabe!)
- Es gibt vordefinierte Templates: Textinhalte werden immer ausgegeben (außer, man unterdrückt das ausdrücklich)
- Die Elemente im Ausgangsdokument gehören einem Namensraum an, dieser wird aber in den XPath-Ausdrücken nicht berücksichtigt (keine Ausgabe!)

i|d|e

XML

Kurzreferenz für

Einsteiger

(mit DTD, XPath und XSLT)

Institut für Dokumentologie und Editorik

www.i-d-e.de
info@i-d-e.de

XML – Extensible Markup Language

XML ist eine „Auszeichnungssprache“, oder genauer gesagt, eine „Metasprache“, nach deren Regeln einzelne Auszeichnungssprachen entworfen werden können. Diese konkreten Sprachen bilden z.B. ein Modell einer Textsorte oder eines Wissensbereichs und erlauben die elektronische Codierung von Texten und anderen Informationsbeständen.

Beispiel

```
❶ <?xml version="1.0" encoding="UTF-8"?>
❷ <!DOCTYPE gedicht SYSTEM "Dateiname.dtd">
❸ <?xml-stylesheet type="text/xsl"
                   href="Dateiname.xsl"?>
❹ <gedicht❺xmlns="URI">
❻   <titel❼lang=❽"de">Kreti❾&amp; Pleti
    </titel>
❿   <strophe>...⓫<zeilenwechsel/>...</strophe>
⓬   <!-- Kommentar -->
  </gedicht>
```

❶ Jedes XML-Dokument beginnt mit einer „XML-Deklaration“. Der vom W3C verabschiedete aktuelle XML-Standard ist seit 1998 und immer noch in der Version 1.0

❷ Eine (optionale) Dokumenttyp-Deklaration (DTD-Verknüpfung) ist ein Beispiel für die Verknüpfung eines XML-Dokuments mit einem Schema - einer „Grammatik“, welche die Struktur des Dokuments bestimmt und kontrolliert

❸ Eine (optionale) Verarbeitungsanweisung („processing instruction“) gibt an, wie das Dokument verarbeitet werden soll - hier z.B. durch eine XSLT-Transformation

❹ Ein Element mit dem Namen „gedicht“, hier zugleich das „Wurzelelement“ des Dokuments, das alle anderen Elemente einschließt

❺ Eine (optionale) Namensraum-Deklaration (xml namespace)

❻ Ein Element mit dem Namen „titel“ mit Inhalt, hier: Zeichendaten

Ein Attribut aus ❼Attributname und ❽Attributwert

❾ Eine Entity (`&entityname;`), hier: `amp=ampersand="&"`

❿ Ein Element mit dem Namen „strophe“ mit Inhalt vom Typ „mixed content“ (Zeichendaten UND weitere Elemente)

⓫ Ein leeres Element (Name = „zeilenwechsel“)

⓬ Ein Kommentar, der bei der weiteren Verarbeitung des Dokuments nicht berücksichtigt wird

Terminologie

- Ein XML-Dokument, das die grundlegenden XML-Regeln erfüllt, ist „wohlgeformt“ (well-formed)
- Ein XML-Dokument, das die Regeln eines Schemas erfüllt, ist „gültig“ (valid)

DTD – Document Type Definition

DTD ist ein (!) Beispiel für eine (ältere) Schemasprache. Weitere (modernere) Schemasprachen sind z.B. XMLSchema (XSD) oder RelaxNG. Ein Schema beschreibt das Vokabular und die Regeln einer Auszeichnungssprache (ihre „Grammatik“). Es legt fest, welche Elemente (und Attribute) in welcher Weise vorkommen können. Das „Inhaltsmodell“ eines Elements gibt an, welche Elemente in welcher Reihenfolge und welcher Häufigkeit in ihm enthalten sein können. Außerdem können Datentypen und mögliche Werte für Attribute festgelegt werden.

Beispiel

```
<!ELEMENT gedicht (titel?, strophe+)>
<!ELEMENT titel (#PCDATA)>
<!ATTLIST titel lang (de|en|fr) #REQUIRED
                n CDATA #IMPLIED>
<!ELEMENT strophe (#PCDATA|zeilenwechsel)*>
<!ELEMENT zeilenwechsel EMPTY>
```

Allgemein:

```
<!ELEMENT elementname (Inhaltsmodell)>
<!ATTLIST elementname attributname
          (attributwerte) #Verbindlichkeit>
```

DTDs haben den Vorteil einer kompakten Syntax. Im Vergleich dazu können andere Schemasprachen etwas unübersichtlich sein, haben aber andere Vorteile: XML Schema z.B. ist ausdrucksmächtiger, weil es eine genauere Festlegung von Inhaltsmodellen und Datentypen erlaubt. Außerdem ist es selbst wieder XML und kann somit auch mit XPath und XSLT verarbeitet werden. Es ist modular aufgebaut und erweiterbar, was bei DTDs nicht der Fall ist.

XML-Regeln

- Elemente müssen geschlossen werden: `<elementname>Inhalt</elementname>` (Öffnendes und schließendes Tag oder `<elementname/>` (Leeres Element)
- Alle Elemente müssen sauber geschachtelt sein (keine Überlappung! **Ein** Wurzelelement! Die Schachtelung führt zu einer Baumstruktur)
- Element- und Attributnamen unterscheiden Groß- und Kleinschreibung
- Gleiche Attribute dürfen sich in einem Element nicht wiederholen; Attributwerte stehen in Anführungszeichen
- Entities z.B. für `<` (`<`), `>` (`>`), `&` (`&`), `'` (`&apos;`), `"` (`&quot;`) - man kann Entities auch selbst definieren
- Elemente können bestimmten Namensräumen angehören (werkzeug:mutter ≠ familie:mutter); Notation: `<namensraum:elementname>`

DTD-Regeln

- Abfolge: `,` bedeutet festgelegte Reihenfolge („dann“)
  `|` bedeutet Auswahl aus Alternativen („oder“)
- Quantoren (bei Elementen, Gruppen, Modellen):
  nichts = genau einmal
  `?` = kein- oder einmal
  `+` = ein oder mehrere Male
  `*` = kein, ein oder mehrere Male
- Zeichendaten in Elementen: `#PCDATA`
- Mixed Content: `(#PCDATA|elementname)*`
- Gruppen in Inhaltsmodellen durch `(...)`
- Verbindlichkeit von Attributen:
  `#IMPLIED` = kann vorkommen
  `#REQUIRED` = muss vorkommen
  `#FIXED` = fester Wert
- Attributwerte nicht festgelegt: `CDATA`
- Kommentar: `<!-- Text -->`

Einige wichtige XPath-Funktionen

not(XPath)
– kehrt den Wahrheitswert des XPath-Ausdrucks um

name() bzw. ***local-name()***
– Name des aktuellen Knotens

count(XPath)
– Anzahl der Knoten in einer Knotenmenge

position()
– Position eines Knotens im Dokument; abgekürzte Syntax als Prädikat: `[n]`

string(object?) bzw. ***number(object?)***
– konvertiert etwas explizit in eine Zeichenkette/eine Zahl

string-length(string?)
– Länge der Zeichenkette

concat(string,string,string*)
– Vernüpfung mehrerer Strings

contains(string,string)
– enthält der erste den zweiten String?

substring(string,number,number?)
– extrahiert aus einem String ab einer bestimmten Position (1. Zahl) eine bestimmte Menge Zeichen (2. Zahl)

translate(string,string,string)
– ersetzt im ersten String alle Zeichen des zweiten Strings durch das Zeichen an der gleichen Stelle im dritten

normalize-space(string?)
– entfernt Leerzeichen am Anfang und Ende und ersetzt mehrere Leerzeichen durch eines

→ Weitere Funktionen:
http://www.w3schools.com/xpath/xpath_functions.asp

XPath – XML Path Language

XPath ist eine Sprache zur Navigation in XML-Dokumenten und zur Adressierung, Selektion und Rückgabe von Knoten/Knoten-Sets/Bäumen und Strings/Zahlen/Wahrheitswerten (Booleans) sowie Sequenzen als Reihen dieser Dinge.

XPath-Ausdrücke setzen sich aus einzelnen Pfadangaben (Lokalisierungsschritten) zusammen, die Achsen (`Achse::`), Knotentests (`elementname`), Bedingungen (`[...]`), Funktionen (`function()`) und andere Anweisungen enthalten können.

Pfadangaben beziehen sich immer auf einen „Kontext“. Das ist die Stelle des Baumes, an der man sich aktuell befindet. Lokalisierungsschritte folgen vertikalen Achsen (der Baumstruktur/Hierarchie, mit Attributen) oder horizontalen Achsen (Reihenfolge im Dokument).

Beispiel

`//*`
– beliebige Elemente in beliebiger Tiefe

`/gedicht/strophe/zeilenwechsel`
– Pfad zu Element `zeilenwechsel` als Kind von `strophe` als Kind von `gedicht` als Kind des aktuellen Kontextes

count(//gedicht/*)
– Anzahl aller Kinder von allen Elementen `gedicht`

`/gedicht/titel[@lang!="de"][@lang!="en"][1]`
– von allen `titel`, die Kind von `gedicht` sind und deren Attribut `lang` nicht den Wert `"de"` oder `"en"` hat, das erste (***position()*****`="1"`**)

/titel[contains(.,"Kreti") or contains(.,"Pleti")]
– alle Elemente `titel`, die die Zeichenkette `"Kreti"` oder `"Pleti"` enthalten

`//index[`***not(***`preceding::index=.)]`
– alle Elemente `index`, zu denen es kein vorhergehendes Element `index` mit dem gleichen Inhalt gibt (nur das erste von gleichen)

Kurz	Achse	Bedeutung
`.`	`self`	der aktuelle Kontextknoten
nichts	`child`	direkt untergeordnete Knoten
`..`	`parent`	direkt übergeordnete Knoten
	`ancestor`	übergeordnete Knoten
	`ancestor-or-self`	dito, plus Kontextknoten
	`descendant`	untergeordnete Knoten
`//`	`descendant-or-self`	dito, plus Kontextknoten
	`following`	nachfolgend im Dokument
	`following-sibling`	nachfolgend, gleicher parent
	`preceding`	vorhergehend im Dokument
	`preceding-sibling`	vorhergehend, gleicher parent
`@`	`attribute`	Attributknoten