

Web- und Multimediabasierte Informationssysteme

Vorlesungsmitschrieb

des Studiengangs

Informationstechnik

von

Jan Ulses

15. September 2014

Dozent:	Jürgen Röthig
E-Mail:	jr@roethig.de
Vorlesungszeitraum:	29.09.14 - 31.03.14
Klausurtermin:	19.12.2014
Autor:	Jan Ulses
Kurs:	TINF12B3
Ausbildungsfirma:	Harman/Becker Automotive Systems GmbH
Studiengangsleiter:	Jürgen Vollmer

Inhaltsverzeichnis

1	XML	5
1.1	Beispiele für XML-basierte Sprachen	5
1.2	DOCTYPE-Deklaration	6
1.3	Wesentliche Eigenschaften von XML-Dateien	6
1.4	Document Type Definition	6
1.5	ELEMENT-Deklaration	7
1.6	ATTLIST-Deklaration	8
1.6.1	Beispiel	10
2	XSL	11
2.1	Bestandteile	11
3	XSLT	12
3.1	Transformation	12
4	XPath	14
4.1	XPath Baumstruktur	14
4.2	Lokalisierungsschritte	15
4.2.1	Achsenausdrücke (ausführliche Notation)	16
4.2.2	Knotentest	16
4.2.3	Prädikate	16
	Abbildungsverzeichnis	18
	Tabellenverzeichnis	19
	Listings	20

KAPITEL 1

XML

- eXtensible Markup Language
- Mittel um konkrete Auszeichnungssprachen zu definieren
- XML selbst ist eine Metasprache, keine eigene (konkrete) Sprache

Auszeichnungssprache: Sprache um reinen Text weitere Eigenschaften mitzugeben

Designorientiert: Textbestandteile bekommen Aussehen (z.B. Fettdruck, rote Farbe). Beispiel: klassisches Word 1990, Druckformate (PostScript, PCL)

Strukturorientiert: Struktur oder spezielle Funktion (z.B. Überschrift, Absatz, Liste, Tabelle). Beispiel: HTML, LaTeX, SGML, die meisten XML-basierten

1.1 Beispiele für XML-basierte Sprachen

- XHTML (auch HTML5 in XML-Variante)
- SVG (Grafikformat)
- XSD (Sprache zur Definition XML-basierter Sprachen)
- Viele Konfigurationsdateien vieler Software-Pakete (z.B. Apache)
- Dokumentformate (aktuellere) von Microsoft Office (.docx) oder Open Office
- Austauschformate für Inhalte von relationalen Datenbanken

1.2 DOCTYPE-Deklaration

```
<!DOCTYPE html(root Tag) | PUBLIC(bzw. SYSTEM, PRIVATE) "Public-Id"(bei  
PUBLIC) "Syst-Id"(nicht bei PRIVATE)>
```

Listing 1.1: Syntax einer DOCTYPE-Deklaration

Public-ID: ungefähr wie bei HTML “//W3C/DTD/XHTML1.1/EN“

System-ID: URL, verweist auf konkrete Grammatik in Form einer DTD

1.3 Wesentliche Eigenschaften von XML-Dateien

Es gibt zwei wesentliche Eigenschaften, welche jedes XML-basierte Dokument erfüllen muss bzw. sollte.

- Wohlgeformtheit (z.B. XML-Deklaration)
 - Wurzel-Tag, welcher das komplette Dokument umschließt
 - Tags paarweise, also Start- und Endtag
 - Korrekte Schachtelung (letzter geöffneter Tag zuerst schließen)
- Gültigkeit (z.B. DOCTYPE-Deklaration, insbesondere Verweis auf DTD)
 - Entspricht einer konkreten Grammatik (Tagnamen, Attributnamen und Zugehörigkeit, Enthaltenseinsmodell (Inhalt eines Tags))

1.4 Document Type Definition

Document Type Definitionen (kurz: DTD)

- Definiert eine konkrete Grammatik (XML-basiert)
- Besteht aus Text
- Besteht nur aus Deklarationen (wegen fehlendem Wurzeltag nicht XML-basiert)

1.5 ELEMENT-Deklaration

```
<!ELEMENT tagname inhaltsmodell>
```

Listing 1.2: Syntax einer ELEMENT-Deklaration

tagname: Name des Elements/Tags bestehend aus Buchstaben (Klein- und Großschreibung wird unterschieden, <bla> ist nicht gleich <bLa>) und Ziffern, 1. Zeichen muss Buchstabe oder Unterstrich sein, theoretisch beliebig lang, praktisch kleiner 256 Zeichen, keine Umlaute verwenden.

inhaltsmodell:

EMPTY	(Bsp. aus XHTML: <!ELEMENT br EMPTY> für leere Tags ohne Inhalt)
(#PCDATA)	für beliebige Zeichenfolgen (außer „<“, „>“, „&“ und „““) insbesondere keine Tags z.B. <!ELEMENT title (#PCDATA)>
sequenz	z.B. (tagname1, tagname2) → Abfolge von tagname1 und tagname2 z.B. <!ELEMENT html (head, body)>
auswahl	z.B. (tagname1 — tagname2)
gemischt	((#PCDATA) — auswahl)*

Alle Inhaltsmodelle können mit nachgestellten Quantoren versehen werden:

- (inhaltsmodell)* beliebig oft (inkl. Keinmal)
- (inhaltsmodell)+ beliebig oft, aber mindestens einmal
- (inhaltsmodell)? einmal oder keinmal

Entitäten:

<	„<“	Less than
>	„>“	Greater than
&	„&“	Ampersand
"	„““	Quotation mark
ä	„ä“	
Ä	„Ä“	

1.6 ATTLIST-Deklaration

```
<!ATTLIST tagname attrname attrtype voreinstellung (optional) >
```

Listing 1.3: Syntax einer ATTLIST-Deklaration

attrname: Name des Attributs, genauso aufgebaut wie Tagnamen

attrtype:

CDATA	beliebige Zeichenfolgen inklusive „<“ und „>“, Einschränkung bei einfachen/doppelten Anführungszeichen
ID	dokumentweit eindeutiger Wert, Einschränkung an Werteraum wie bei Tag- und Attributnamen! d.h. z.B. Zahlenwerte sind keine gültigen Werte vom Typ ID! Beispiel aus HTML: <!ATTLIST a id ID>
IDREF	Referenz/Verweis auf einen ID-Wert, Einschränkung der Werte wie oben, aber keine Eindeutigkeit gefordert, da beliebig oft auf denselben ID-Wert verwiesen werden darf
IDREFS	beliebig viele ID-Werte, getrennt durch Leerzeichen
NMTOKEN(S)	„Name“, d.h. Zeichenfolge von beliebig vielen Buchstaben, Ziffern, manchen Sonderzeichen (insb. aber kein Leerzeichen), auch das erste Zeichen darf ein beliebiges Zeichen der Zahlenmenge sein bzw. mehrere Namen durch Leerzeichen getrennt aufzählung: (nmtoken1—nmtoken2—nmtoken3—...) der Attributwert kann nur einer der aufgeführten „Namen“ sein
ENTITY	Verweis auf Entitäten → externe (auch binäre) Daten
ENTITIES	Verweis auf Entitäten → externe (auch binäre) Daten
NOTATION	Daten mit spezieller Interpretation

voreinstellung:

#REQUIRED	Pflichtattribut
#IMPLIED	optionales Attribut
#FIXED	wert, Attribut mit festem Wert wert
wert	#IMPLIED-Attribut mit Default-Wert wert
[fehlt]	#IMPLIED-Attribut ohne Default-Wert

Die „Gültigkeit“ einer XML-Datei kann anhand der DOCTYPE-Deklaration und der darin referenzierten DTD überprüft werden → mittels einem Validator z.B. für HTML-Dateien "<http://validator.w3.org/>". Problem: Zugriff des Validators auf die DTD? Muss die DTD auf einem öffentlich zugänglichen WebSpace liegen? → NEIN, Abhilfe: Inline-DTD, siehe das Beispiel aus Listing 3.1 auf Seite 12.

```
<?xml version="1.0" ?>
2 <!DOCTYPE bla [
    <!ELEMENT ...>
4     .
    .
6 <!-- ATTLIST ... -->
    .
8     .
] >
10 <bla>
    .
12     .
</bla>
```

Listing 1.4: Inline-DTD Beispiel

1.6.1 Beispiel

Name	Vorname	Matrikelnummer	Kursbezeichnung	Wahlfach
Müller	Max	012345	TINF12B3	WuMBasis
Maier	Moritz	4711	TINF12B3	Shit
Schulze	Marta	0815	TINF12B5	Gaming

```

<studis>
2  <studi matrikelnummer="012345">
    <name nach="Mueller" vor="Max" />
4    <kurs> TINF12B3 </kurs>
    <wahlfach> WuMM </wahlfach>
6  </studi>
    <!-- entsprechend fuer Maier und Schulze -->
8 </studis>

```

Listing 1.5: Listeneinträge

```

<!ELEMENT studis (studi)*>
2 <!ELEMENT studi (name, kurs, wahlfach)>
<!ELEMENT name EMPTY>
4 <!-- ATTLIST name nach CDATA REQUIRED attrtype evtl. NMTOKEN vor CDATA
    REQUIRED attrtype evtl. NMTOKENS -->

```

Listing 1.6: Baumerstellung per !ELEMENT

Anzeige abstrakter XML-Daten (nicht HTML oder SVG) im XML-fähigen WebBrowser?

- strukturierte Liste (mit Einschränkungen, Elemente aus- und einklappbar)
- nicht besonders anschaulich
- kann mit CSS deutlich „aufgehübscht“ werden
- bessere Variante: XSLT (XML Style Sheet Language Transformation)
Achtung: Trotz des Namensbestandteils “Stylesheet macht eine XSLT viel mehr als nur Aussehen festzulegen!

KAPITEL 2

XSL

2.1 Bestandteile

Die XML Stylesheet Language besteht aus:

- XSLT: XSL Transformation, Sprache zur Transformation von „XML-Konstrukten“ in andere XML-Konstrukte (oder auch „Konstrukte“ in textbasierten Sprachen)
- XPath: XML Path Language, Sprache zur Auswahl von spezifischen „XML-Konstrukten“ aus der XML-Quelldatei
- XML-FO: XML-Formatting Objects, spezielle XML-basierte Sprache zur layoutgetreuen Ausgabe (nicht struktur- sondern designorientierte Sprache)
- im Folgenden für uns in der Vorlesung interessant: XSLT, XPath nicht jedoch XML-FO

Bsp. für Anwendung: XSLT zur Wandlung der abstrakten „Studis-Datei“ in eine HTML-Datei mit entsprechender Tabelle der Studis

Wer führt die Transformation durch?

- *standalone-Tool*: XSLprocessor (in gängigen Linux-Distributionen enthalten) Apache xalan saxon (von Michael Kay) (unterstützt auch XSLT Version 2)
- *serverseitig*: Integration der XSLT in einem WebServer, d.h. der WebServer liefert bei Anforderung der XML-Datei bereits die mittels XSLT transformierte Datei aus! Apache Project Cocoon Perl-Modul AxKit
- *clientseitig*: integriert in gängige WebBrowser → Mozilla Firefox, MS Internet Explorer, Opera, Chrome, Safari können XSLT!

KAPITEL 3

XSLT

3.1 Transformation

Werkzeug zur Transformation von XML-basierten Daten in (meist andere) XML-basierte Daten.

```
<?xml version="1.0" ?> <!--Hinweis: Attribut encoding="UTF-8" ist bei
XML default-->
2 <!DOCTYPE studis SYSTEM "url/zur DTD"> //<?+<! sind Deklarationen wobei
  <! auf > endet.
  [KEINE DOCTYPE-Deklaration!]
4 <xsl:stylesheet
  version="1.0" //version->Namensraumdeklaration fuer XSLT, Praefix->
    Postfix
6   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="Namespace der Ausgabesprache, z.B. HTML" //
     Namensraumdeklaration fuer Ausgabesprache, Verwendung ohne Postfix
     und Praefix (Grund: Ersparung von Schreibarbeit)
8 >
   <xsl:output method="xml" encoding ="UTF-8" //method->auch html( bitte
     nicht angeben!) oder text
10  doctype-public"... " //Public Doctypes (Doctype definiert den
    HTML-Standart)
    doctype-system"url/zur/DTD" //fuer system Doctype deklaration
12  <!--Template fuer die Definition der Transformation-->
</xsl:stylesheet>
```

Listing 3.1: Definition einer XML-Datei zur Transformation

```
<?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE studis SYSTEM "url/zur/DTD">
```

Listing 3.2: Transformierte XML-Datei

Transformationsvorschriften in Form von Templates (Schablonen)

- Templates werden nacheinander notiert, d.h. sie können nicht geschachtelt werden.
- Templates ersetzen irgendwelche Knoten (Tags und Attribute) aus der Quelldatei.

```
<xsl:template match="XPath-Ausdruck">
2 <!--(wohlgeformte) Ausgabe des des Templates, also Text, Tags(inklusive
   Attribute) und weitere Verarbeitungsanweisungen-->
</xsl:template>
```

Listing 3.3: Syntax einer xsl:template-Deklaration

KAPITEL 4

XPath

4.1 XPath Baumstruktur

Sprache zur Auswahl bestimmter Knoten eines XML-Dokuments. Meist relativ zur aktuellen Position im XML-Dokument.

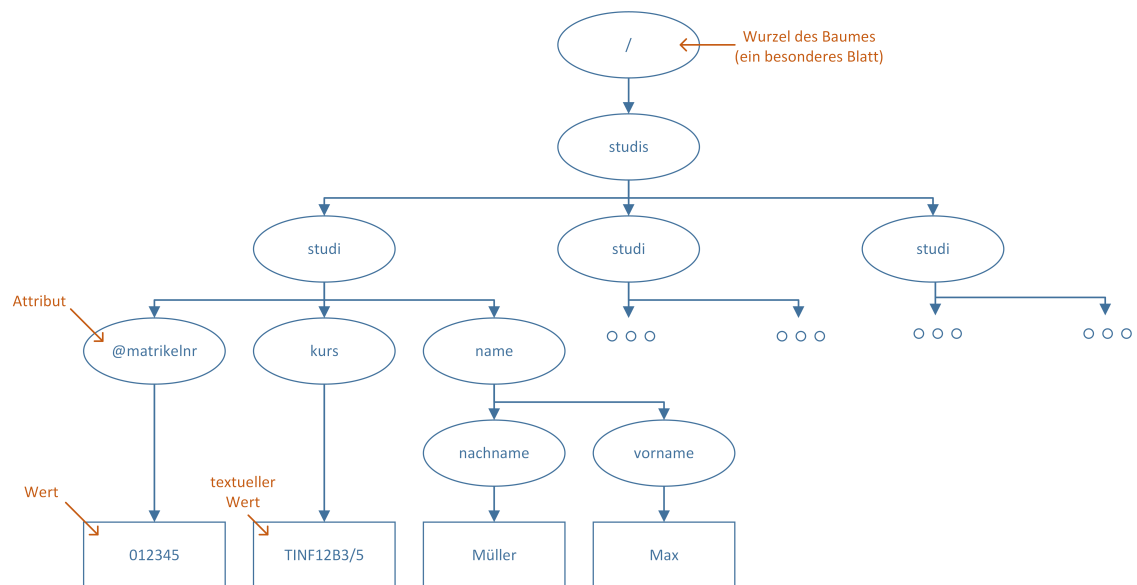


Abbildung 4.1: Baumdarstellung einer XML-Datei

Beschreibung der Knotensyntax:

„.“ → aktueller Knoten

„..“ → Elternknoten

„tagname“ → Kindelement mit „tagname“

„@attrname“ → Attribut mit attrname

„text()“ → Textknoten

„/“ → Wurzel

„//“ → irgendwo im Baum

mehrere Lokalisierungsschritte werden durch „/“ verbunden nacheinander angegeben. Bsp.: /studis/studi/name/nachname/text()

XPath Ausdrücke liefern im allgemeinen eine Knotenmenge, d.h. mehrere Knoten (oder auch keinen)

4.2 Lokalisierungsschritte

bisher: „verkürzte Notation“

außerdem: ausführliche Notation

axis::nodetest[predicate]

(predicate ist optional)

4.2.1 Achsenausdrücke (ausführliche Notation)

root	Wurzelknoten	„/“
child	Kindknoten	„/“ (nicht am Anfang bzw. weglassen)
parent	Elternknoten	„..“
self	aktueller Knoten (Kontextknoten)	„.“
ancestor	Vorfahren, übergeordnete Knoten (Eltern, Großeltern,...)	
descendent	Nachkommen, untergeordnete Knoten (Kinder mit Kindeskindern)	
ancestor-or-self	Vorfahren inkl. Kontextknoten	
descendent-or-self	Nachkommen inkl. Kontextknoten	
following	nachfolgende Knoten (ohne Kinder und Kindeskindern des Kontextknotens)	
following-sibling	nachfolgende Geschwisterknoten (d.h. nachfolgende Knoten mit demselben Elternknoten wie der Kontextknoten)	
preceding	vorhergehende Knoten	
preceding-sibling	vorhergehende Geschwisterknoten (d.h. vorhergehende Knoten mit demselben Elternknoten wie der Kontextknoten)	
attribute	Attributknoten	„@“

4.2.2 Knotentest

- Knotenname/tagname/attrname
- „*“ als Joker für beliebige Knotennamen
- text(), comment() für Text- bzw. Kommentarknoten

4.2.3 Prädikate

Prädikate stehen immer in eckigen Klammern: „[Prädikatausdruck]“

- Zahl: Nummer des Knotens, Nummerierung beginnt bei 1
- Vergleich: z.B. = [@farbe = "blau"] weitere: !=, >, <, >=, <=
- numerische Operatoren: +, -, *, div, mod (alles Ganzzahloperatoren)
- knotenmengen Funktionen: count (...) Anzahl der Elemente

zurück zu:

```

1 <xsl:template match="studis">
2   <html>
3     <head>
4       <title>Studis an der DHBW</title>
5     </head>
6     <body>
7       <table><xsl:applytemplates /></table>
8     </body>
9   </html>
10 </xsl:template>

12 <xsl:template match="studi">
13   <tr>
14     <td><xsl:value-of select="name/nachname/text()" /></td>
15     <td><xsl:value-of select="name/vorname" /></td>
16   </tr>
17 </xsl:template>

```

Listing 4.1: Praktisches Beispiel für xsl:template

xsl:value-of Syntax

```
<xsl:value-of select="XPath-Ausdruck" />
```

Listing 4.2: xsl:value-of Syntax

liefert den textuellen Wert eines Knotens bzw. einer Knotenmenge zurück textueller Wert:

- eines Textknotens \Rightarrow Text selbst eines Attributknotens \Rightarrow Wert des anhängenden Textknotens
- eines Elementknotens (eines „Tags“) \Rightarrow Konkatanation der Werte aller Elemente und Textknoten, welche Kinder des Elementknotens sind

xsl:apply-templates Syntax

```
<xsl:apply-templates select="XPath-Ausdruck" />
```

Listing 4.3: xsl:apply-templates Syntax

- sucht abhängig vom Kontextknoten nach weiteren passenden Templates und führt diese aus (für Kindelemente, kann weiter eingeschränkt und auch ausgeweitet werden über optionales select-Attribut mit XPath-Ausdruck)

Abbildungsverzeichnis

4.1	Baumdarstellung einer XML-Datei	14
-----	---	----

Tabellenverzeichnis

Listings

1.1	Syntax einer DOCTYPE-Deklaration	6
1.2	Syntax einer ELEMENT-Deklaration	7
1.3	Syntax einer ATTLIST-Deklaration	8
1.4	Inline-DTD Beispiel	9
1.5	Listeneinträge	10
1.6	Baumerstellung per !ELEMENT	10
3.1	Definition einer XML-Datei zur Transformation	12
3.2	Transformierte XML-Datei	13
3.3	Syntax einer xsl:template-Deklaration	13
4.1	Praktisches Beispiel für xsl:template	17
4.2	xsl:value-of Syntax	17
4.3	xsl:apply-templates Syntax	17