XML

XML ist eine Auszeichnungssprache (Extensible Markup Language) und dient der strukturierten Darstellung von Daten. Wie HTML verwendet XML auch Tags zur Auszeichnung von Elementen. Im Gegensatz zu HTML, beschreiben Tags nicht wie der Inhalt dargestellt werden soll, sondern wie der Inhalt beschrieben werden kann. XML dient dazu, Struktur, Inhalt und Darstellung eines Dokuments stark zu trennen.

Tags sind durch spitze Klammern gekennzeichnet. Unterschieden wird zwischen öffnenden Tabs, z.B. und schließenden Tabs, z.B. . Jedes geöffnete Tab muss auch wieder geschlossen werden.

Da XML wesentlich allgemeiner ist als HTML, ist es möglich weitere Markup Sprachen zu definieren. So lässt sich mit XML, die Markup Sprache HTML definieren.

Ein XML-Dokument kann so aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<buch>
  <titel>Der Traum</titel>
  <autor/>
</buch>
```

1 Aufbau eines XML-Dokuments

Ein XML-Dokument enthält Deklarationen, Elemente, Attribute, Kommentare, Zeichenreferenzen und Verarbeitungsanweisungen. Im Folgenden werden die wichtigsten dargestellt.

- Deklaration Leitet ein XML-Dokument ein:
  - xml kennzeichnet das Dokument als XML
  - version="1.0" Versionsnr. des verwendeten XML-Standards
  - encoding="utf8" Festlegung der Zeichenkodierung
  - standalone="yes" gibt es eine externe Auszeichnungsdeklaration?
- XML-Elemente

Die wichtigsten Einheiten, in die sich ein Dokument gliedern lässt, werden Elemente genannt. Das Wurzelelement umschließt alle weiteren Elemente, wodurch sich hierarchisch eine Baumstruktur ergibt. Die Bedeutung der Elemente werden in den Tags beschrieben. (Siehe Vonhoegen(2011), Seite 49.)

- XML-Attribute

Alle Elemente können mit beliebigen Attributen versehen werden, die bestimmte Eigenschaften eines Elements festhalten sollen. Attribute werden nur im Starttag des Elements platziert. Der Attributname darf im selben Element nur einmal vorkommen.

`<haus nr="22" nr="23">` ist **nicht** zulässig

Mehrere Attribute und ihre Werte eines Elements werden durch Leerzeichen getrennt.

```
<person vorname="Jane K" nachname="Austen">
```

Auch leere Elemente wie `<leer\>` können Attribute enthalten:

```
<leer name="leeres Element"\>
```

Während Elemente andere Elemente enthalten können, dürfen Attribute nicht verschachtelt werden.

- XML-Kommentare

Grundsätzlich lassen sich Kommentare im gesamten Dokument einfügen, außer innerhalb von Deklarationen und Tags. Auch darf kein Kommentar vor der XML-Deklaration stehen. Ein Kommentar beginnt mit .

- Zeichenreferenzen

XML bietet bestimmte XML spezifische Zeichen an. Um die Zeichen zwischen Tags dennoch benutzen zu können, ohne dass diese vom XML Prozessor als Markupzeichen interpretiert werden, müssen diese mit bestimmten Zeichenreferenzen ersetzt werden. Hierfür stellt XML fünf Entitäten bereit:

- `&` für das & (Et-Zeichen)
- `>` für das <(Größer-als-Zeichen)
- `<` für das >(Kleiner-als-Zeichen)
- `&quot;` für das "(Ersatzzeichen für Anführungszeichen)
- `&apos;` für das ' (Ersatzzeichen für Apostroph)

  In DTD-Dokumenten können auch eigene Entitäten deklariert werden. (Entitäten sind spezielle Zeichenfolgen, die von einer Anwendung, die das XML-Dokument bearbeitet, durch einen bestimmten Text ersetzt werden. Entitätsnamen beginnen stets mit dem Symbol & und enden mit ;)

2 Anforderungen an ein XML-Dokument

XML-Dokumente müssen *wohlgeformt* sein, d.h. folgende Regeln müssen erfüllt sein:

1. Es gibt nur *ein* Wurzelelement (Tag welche alle weiteren Tags umschließt)

```
<buch>
<titel> Der Traum </titel>
<autor> Thomas Mann </autor>
</buch>
```

2. Jedes Element muss einen Start- und einen Endtag besitzen

```
<buch> Informationen zum Buch
  <autor> Thomas Mann </autor>
  <titel> </titel>
  <abstract/>
</buch>
```

Falsch:

```
<buch> Informationen zum Buch
  <autor> Thomas Mann
</buch>
```

3. Elemente dürfen geschachtelt sein, sich aber nicht überlappen

Thomas Mann <\beispiel><\autor> **Falsch**

Thomas Mann <\autor><\beispiel> **Richtig**

4. Attributw erte müssen in Anführungszeichen stehen

<\beispiel>

5. Namensgebung:
   (a) Elementnamen beginnen mit `[A-Za-z\:\_]`
   (b) "xml" darf nicht im Namensanfang stehen
   (c) Namen sind casesensitive: ungleich <\beispiel>

3 Namensräume

Aufgrund der freien Namensw ahl in XML von Element- und Attributnamen, kann es zu Überschneidungen von Namen mit unterschiedlichen Bedeutungen kommen. Im Hinblick darauf, dass XML-Dokumente von unterschiedlichen Programmen ausgew ertet und bearbeitet w erden können, führen Mehrdeutigkeiten schnell zu unerw ünschten Ergebnissen. Als Beispiel dient der einfache Elementname

Beispielsw eise kann die Namensraumdefinition im Element so aussehen:

```
<buch xlns:ttl=http://titel.com/buch>... <\buch>
```

- xlns ist der reservierte Attributname für Namensräume
- ttl das selbst gew ählte Präfix als Abkürzung für den Namensraum
- http://titel.com/buch der eigentliche Namensraum Das Prefix w ird dann dem betroffenen Element oder Attribut zur Eindeutigkeit vorangestellt

```
<ttl:titel>.. <\ttl:titel>
```

4 DTD

Eine XML-Dokumenttyp-Definition(DTD) definiert die Regeln für ein XML-Dokument. Sie definiert w elche Elemente (und Attribute) es gibt, und w ie diese im Dokument angeordnet w erden müssen. Ein XML-Dokument w ird gültig genannt, w enn es alle Regeln seiner DTD befolgt.

- Dokumenttyp-Deklaration

Die DTD w ird durch eine Dokumenttyp-Deklaration in das XML-Dokument eingebunden. Sie teilt dem Prozessor mit, w o die DTD zu finden ist. Ist die DTD extern, erfolgt die Einbindung über:

```
<!DOCTYPE wurzelelement SYSTEM "Document-Name.dtd">
```

Intern lässt sich die DTD im XML-Dokument selbst definieren als:

```
<!DOCTYPE wurzelelement [
<!ELEMENT buch(autor, titel)>
<!ELEMENT autor>
.. ]>
```

- Elementdeklaration

Elemente werden über eine Deklaration eingeführt. In der Deklaration folgt der Name des Elements, wie er auch in der XML Datei vorkommen soll. Dahinter folgt in einem Klammerpaar ein Inhaltsmodell.
Beispiel

```
<!ELEMENT bild (titel, url, abstract, person+)>
<!ELEMENT titel(#PCDATA)>
[...]
```

Inhaltstyp	Beispiel	Bedeutung
EMPTY		ein Element ohne Inhalt, aber evtl. mit Attributen
ANY		beliebiger Inhalt, solange wohl geformtes XML
#PCDATA		Das Element enthält nur Zeichendatan
Gemischter Inhalt		Das Element enthält nur Zeichendaten oder eine Kombination von Zeichendaten und Unterelementen
Elementinhalt		Ein Element enthält nur Unterelemente
Tabelle 1: Inhaltstype für Elemente		

Die Anzahl der (Unter)Elemente können durch Indikatoren +,*, ? bestimmt werden. z.B. kann das Childelement einmal, keinmal oder mehrmals vorkommen.

- Attributlistendeklaration

Diese Deklaration führt alle Attribute, die in einem gegebenen Element verwendet werden können. Sie wird durch

Attributtyp	Beschreibung
CDATA	Einfache Zeichendaten ohne Markupzeichen
ENTITY	Name einer in der DTD nicht deklariertennicht geparsten Entität
Entities	Durch Leerzeichen getrennte Liste von Entitä-ten
ID	Eindeutiger XML-Name, der als Identifiziereeines Elements verwendet werden kann
IDREF	Verweis auf den ID-Identifizierer eines Ele-ments
IDREFS	Liste von Verweisen auf ID Identifizierern
NMTokens	Liste von mehreren XML Namen
NOTATION	Verweis auf eine Notation z.B. Name für einnicht XML-Format wie Grafikdatei

Tabelle 2: Attributtypen in DTD

Vorgabedeklaration	Beschreibung
#IMPLIED	Keine Vorgabe
#REQUIRED	Kein Vorgabewert, aber Wert erforderlich

#FIXED Wert	Weist dem Attribut einen Standardw ert zu

Tabelle 3: Werte für die Vorgabedeklaration

Ein DTD-Dokument hat immer als *.dtd* gespeichert.

```
beispiel.dtd
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT mediensammlung ( buch | film | musik )+>
<!ENTITY %katalogdaten "((kuenstler+|autor+|regie+),titel,untertitel?,jahr)">
<!ELEMENT kuenstler (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
<!ELEMENT regie (#PCDATA)>
<!ELEMENT titel (#PCDATA)>
<!ELEMENT untertitel (#PCDATA)>
<!ELEMENT jahr (#PCDATA)>
<!ELEMENT film %katalogdaten;>
<!ELEMENT musik %katalogdaten;>
```

Das eigentliche XML-Dokument bindet die externe DTD *beispiel.dtd* ein:

```
<?xml version="1.0 " encoding="UTF-8" standalone="no" ?>
<!DOCTYPE mediensammlung SYSTEM "example3.dtd">
<mediensammlung>
  <buch>
    <autor>Paul Auster</autor>
    <titel>Die New York-Trilogie</titel>
    <jahr>1989</jahr>
  </buch>
  <film>
     <regie>Richard Kelly</regie>
     <titel>Donnie Darko</titel>
     <jahr>2001</jahr>
  </film>
</mediensammlung>
```

- Validierung der w ohlgeformtheit mit Python

Es gibt eine sehr einfache Möglichkeit ein XML-Dokument mit Pyhton zu validieren. Man verw endet die lxml- Library und parsed die XML- Datei:

```
from lxml import etree
parser = etree.XMLParser(dtd_validation=True)
tree = etree.parse("sample.xml", parser)
```

siehe: http://lxml.de/validation.html

5 XML-Schema

DTD ermöglicht die Wohlgeformtheit eines Dokuments zu prüfen, prüft jedoch nicht den Inhalt der verw endeten Tags. Eine andere Möglichkeit Standards für XML- Dokumente einzurichten ist XML-Schema. Hierbei können die geforderten Inhalte von Tags und Attribute genauer angegeben w erden, auch ohne auf eine seperate Syntax zugreifen zu müssen.

- Schema-Definition

Die Schema-Definition befindet sich als Wurzelelement in einer .xsd Datei. Die meisten Parser unterstützen ein Schema auch als

Alternative zu DTD.

```
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema>
```

Ein XML-Schema wird in einem XML-Dokument über folgende Zeile eingebunden:

```
<wurzelelement xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="SCHEMA.xsd">
```

- Elementdefinition

Jedes Element wird in einer -Deklaration angegeben. Unterschieden werden einfache und verschachtelte Elemente. Erstere, die nur einfachen Inhalt ohne weitere verschachtelte Elemente und Attribute besitzen, benötigen im Schema das Attribut *type*, das den zulässigen Typ des Inhalts angibt.
Die Syntax einer einfachen Elementdeklaration ist:

```
<xs:element name="xxx" type="yyy"/>
```

Häufige Datentypen sind:

- xs:string
- xs:integer
- xs:decimal
- xs:boolean
- xs:date

- xs:time

Beispielsweise geben "xs:string" den Typ für beliebigen Text und "xs:integer" den Typ für ganze Zahlen an .
Auch Attribute werden zu der Klasse der einfachen Elemente gezählt und genauso deklariert:

```
<xs:attribute name="xxx" type="yyy"/>
```

Es ist möglich weitere Eigenschaften wie z.B. Defaultwerte und Erforderlichkeit in der Deklaration zum Attribut hinzuzufügen. Einfache Elemente können keine Attribute haben, da sie sonst als verschachtelte Elemente gelten.

Einfache Elemente und Attribute können in ihren Werten auch eingeschränkt werden. Im folgenden Beispiel wird für das Element "Alter" eine Mindest- und eine Höchstgrenze festgelegt. Weitere Beschränkungsmöglichkeiten bildet Tabelle 4

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Beschränkung	Beschreibung
enumeration	Definiert eine Liste akzeptierter Werte

fractionDigits	Definiert die maximale Anzahl erlaubter Dezimalstellen (=> 0)
length	Definiert die erlaubte Anzahl von Zeichen oder Listenelemente (=> 0)
maxExclusive	Definiert eine obere Grenze nummerischer Werte (der Wert muss unter der Grenze liegen)
maxInclusive	Definiert eine obere Grenze nummerischer Werte (der Wert muss kleiner oder gleich der Grenze sein)
maxLength	Definiert die maximal erlaubte Anzahl von Zeichen oder Listenelemente
minExclusive	Definiert eine untere Grenze für nummerische Werte (der Wert muss über der Grenze liegen)
minInclusive	Definiert eine untere Grenze für nummerische Werte (der Wert muss größer oder gleich der Grenze sein)
minLength	Definiert die minimal erlaubte Anzahl von Zeichen oder Listenelemente (>=0)
pattern	Legt eine exakte Folge von Zeichen fest
totalDigits	Legt fest, wie viele Zahlen erlaubt sind (>0)
whiteSpace	Legt fest, wie Leerzeichen zu händeln sind

Tabelle 4: Beschränkungen für Datentypen

Verschachtelte Elemente besitzen ein -Block. In diesem werden alle Elemente und Attribute als einfache Elemente aufgelistet. Besitzt das *Oberelement* selbst ein Attribut, kann die Deklaration gesplittet erfolgen. Zunächst wird das Oberelement wie ein einfaches Element deklariert und mit einer folgenden selbst definierten Typangabe auf seine ihm untergeordneten Elemente verwiesen:

```
<xs:element name="autor" type="personinfo"/>
  <xs:complexType name="personinfo">
    <xs:sequence>
      <xs:element name="vorname" type="xs:string"/>
      <xs:element name="nachname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
```

- Indikatoren

Indikatoren ermöglichen festzulegen wie Elemente verwendet werden dürfen. Zur Auswahl stehen 7 Indikatoren:

Order indicators
(geben die Reihenfolge der Elemente vor)

- **All** Alle Elemente im complexType-Block tauchen nur einmal auf
- **Choice** Nur eines der Elemente taucht im Block auf
- **Sequence** Die Elemente müssen in der Reihenfolge wie sie definiert wurden, auftauchen
  z.B.

```
<xs:element name="buch">
  <xs:complexType>
    <xs:choice>
      <xs:element name="titel" type="String"/>
      <xs:element name="autor" type="String"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Occurrence indicators
(geben die Anzahl der Elemente an)

- **maxOccurs** legt fest wie oft ein Element maximal vorkommen kann
- **minOccurs** legt fest wie oft ein Element mindestens vorkommen muss
  z.B.

```
<xs:element name="buch">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="titel" type="xs:string"/>
      <xs:element name="autor" type="xs:string" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Group indicators
(ermöglichen es Elemente zu gruppieren, und später auf eine Gruppe zu referenzieren- es erfordert innerhalb der Gruppe einen *Order*-Indikator)

- **Group name** name der Gruppierung
- **attributeGroup** name
  z.B.

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="vorname" type="xs:string"/>
    <xs:element name="nachname" type="xs:string"/>
    <xs:element name="geburtstag" type="xs:date"/>
  </xs:sequence>
</xs:group>
<xs:element name="person" type="personinfo"/>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:group ref="persongroup"/>
    <xs:element name="land" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

6 XSL und XSLT

XSL (Extensible Stylesheet Language) ist ein weiterer zur Verfügung gestellter Standard und besteht aus zwei Komponenten:

- XSL-FO zur Formatierung von XML Daten
- XSLT (XSL Transformation) zur Transformation von XML Daten in andere textbasierte Dateien

Die Hauptanwendung von XSL ist XSLT. Denn XSLT ermöglicht es, XML Daten in HTML umzuwandeln. Dadurch können Daten strukturiert in XML gespeichert und in HTML präsentiert werden.

- Transformation mit XSLT

XML-Elemente sollen in eine andere XML gerechte Sprache transformiert werden. Hierfür sollen Elemente und Attribute in Auszeichnungskonstrukte einer anderen Sprache umgewandelt werden. Aus eigenen XML-Elementen wie z.B. *autor* soll der HTML code *[…]* werden. Um auf XML-Elemente zugreifen zu können, wird die eigene XSLT-Sprache XPath verwendet. Die Umwandlungsregeln werden in einer XSLT-Datei in XML festgelegt. Die XSL-Datei wird dann im eigentlichen XML-Dokument eingebunden.

Wichtige Merkmale von XSLT

- Baumstrukturen als Modelle von XML-Dokumenten
- XSLT Stylesheets definieren die Umwandlung der Eingabebaumstruktur in eine Ausgabebaumstruktur
- Zugriff auf Teile des Eingabebaums mithilfe von Xpath
- Vorlage (Templates) definiert den Ausgabebaum

Prinzip der Transformation

1. XML-Dok. wird vom Parser eingelesen, und als Baum interpretiert
2. XSLT-Dok. wird vom Parser eingelesen, und als Baum interpretiert
3. Beide Bäume werden vom XSLT-Prozessor eingelesen
4. Der neue (Ausgabe-)Baum erhält gewünschte serielle Form

Eine Transformation besteht aus einer Reihe von einzelnen Transformationsregeln, die als "Templates" ("Schablonen") bezeichnet werden. Ein Template besitzt ein auf XPath basierendes Pattern ("Muster") das beschreibt, für welche Knoten es gilt, und einen Inhalt, der bestimmt, wie das Template seinen Teil des Zielbaums erzeugt.

- Die Subsprache XPath

Die Sprache XPath lokalisiert Elemente in XML-Dokumenten. Ihre abgekürzte Syntax ähnelt Ausdrücken, die benutzt werden, um in hierarchischen Filesystemen Pfade zu Dateien und Verzeichnissen zu spezifizieren. Der XPath Ausdruck /stadt/hotels//name zeigt auf alle *name*- Elemente, die als Nachfolger von *hotels*-Elementen im Wurzelelement mit Label *stadt* auftreten. In dieser Syntax steht . für die augenblickliche Position in XML-Baum und .. für das jeweilige Elternelement. Bei der ausführlichen Syntax wird für jeden Schritt, explizit die Richtung angegeben, in der der Baum durchwandert werden soll. Der obige XPath Ausdruck wird dann als */child::stadt/child::hotels/descendant::name* dargestellt. In beiden syntaktischen Formen kann an jeden Schritt eine Folge von Prädikaten angehängt werden, die die Auswahl einschränken. Der obige Ausdruck lässt sich beispielsweise auf *stadt*-Elemente beschränken, die auch *flughafen*-Unterlemente haben:

/stadt[.//Flughafen]/hotels//name
bzw.
/child::stadt[descendant::Flughafen]/child::hotels/descendant::name

Ein XPath-Ausdruck besteht aus einer Folge von Lokalisierungsschritten, die durch / verknüpft sind. Lokalisierungsschritte, die ohne / eingeleitet werden sind absolute, andernfalls relative Ausdrücke. Ein Lokalisierungsschritt besteht aus der Angabe der Achse, dem Symbol ::, einem Knotentest (Labelname, text() oder node()) und einer optimalen Folge von Prädikaten (ein in []-Klammern auftauchender XPath-Ausdruck). (Siehe Meuss(2001), Seite 28.)

In XPath verwendete Achsen: child, descendant-or-self, ancestor-or-self, preceding, preceding-sibling, following, following-sibling, attribute, namespace.

In XPath-Ausdrücken können eine begrenzte Zahl an Funktionen verwendet werden. Dazu zählen arithmetische Funktionen (Addition, Multiplikation, Vergleiche), Boole'sche Funktionen (and, or, nor), Stringfunktionen (concat) sowie Funktionen, die auf Knotenmengen arbeiten (count, position). Der XPath-Ausdruck

/descendant::chapter[count(descendant::section)>7]

selektiert *chapter*-Elemente, die mehr als sieben *section*-Elemente enthalten.

- Aufbau einer XSLT Datei

Ein XSL-Stylesheet beginnt immer mit einer XML-Deklaration und seiner Stylesheetdefinition als Wurzelelement:

Ein XSLT-Programm ist eine freie Abfolge von Template-Regeln, die jeweils in das Element eingepackt sind. Ein Template besteht aus einem Suchmuster und einer oder mehreren Anweisungen im Template selbst.
Ein Template kann wie folgt aussehen:

```
<xsl:template match="//book[@price]">
  <html>
    <body>
      <h2>
        <xsl:value-of select="title"/>
      </h2>
      <xsl:apply-templates select="chapter/title"/>
    </body>
  </html>
</xsl:template>
```

Das *xsl:template*-Element enthält im *match*-Attribut einen XPath-Ausdruck, der die Elemente beschreibt, deren Transformation in dieser Regel beschrieben w erden. Im obigen Fall also ein *book*-Element, das ein *price*-Attribut hat. Der Regelrumpf beschreibt w ie der gesamte Inhalt des *xsl:template*-Elements des bisher erzeugten Ausgabedokuments ergänzt w erden soll. Dies kann statisch, durch direkte Angabe der hinzuzufügenden Elemente, oder dynamisch, durch w eitere XSLT-Anw eisungen, geschehen. Im obigen Beispiel w erden die html-Elemente zunächst statisch erzeugt. (Eine HTML Datei besteht aus einem HTML- Kopf (head) und einem -Körper (body). Wie in XML gibt es öffnende und schließende Tags. Das Wurzelelement ist . Im Abschnitt w ird der eigentliche Inhalt zur Webanzeige definiert. Typische Elemente sind `<h1>` für Überschriften, `<p1>` für Paragraphen usw.) Die Unterelemente *h2* und *body* w erden aber dynamisch erzeugt. Der XSLT-Befehl *xsl:value-of* ergänzt das Zieldokument um die Knoten, die durch den XPath-Ausdruck beschrieben w erden, der im *select*-Attribut des *xsl:value-of*-Befehls angegeben ist. Bei handelt es sich um Anw eisungen, die die Template Regeln dann auch ausführt. Durch den Befehl *xsl:apply-templates* w erden rekursiv alle Elemente, die vom gerade behandelten Element über im *select*-Attribut angegebenen Pfad erreichbar sind, nach w eiteren Regeln gesucht, w elche ebenfalls ausgeführt w erden. Nach Beendigung, w erden alle erzeugten Teildokumente an dieser Stelle eingefügt, also direkt nach dem *h2*-Element. (Siehe Meuss(2002), Seite 34.)

- Von XML zu HTML

Im Folgenden w ird ein XML-Dokument in eine HTML Datei umgew andelt. Das *XML-Dokument* sieht w ie folgt aus:

```
<TEI>
<teiHeader>
<fileDesc>
<title>[DRAFT!] (2009–): Wittgenstein TS 213: Ts–213.xml</title>
<author>Ludwig Wittgenstein</author>
<editor><persName>Alois Pichler</persName>
<orgName ref="http://wab.aksis.uib.no/">Wittgenstein Archives at the University of Bergen (WAB)</orgName>
</editor>
</fileDesc>
</teiHeader>
<text>
<ab n="Ts–213,44r">
<satz n="Ts–213,44r_[1]"> das ist der Satz AAA nnn </satz>
<satz n="Ts–213,44r_[2]"> das ist der Satz BBB nnn <pb facs="Ts–213,45r"/> Don't forget me <lb/>this week–<lb/>end </satz>
</ab>
<ab n="Ts–213,45r">
<satz n="Ts–213,45r_[1]"> das ist der Satz XAAA nnn
</satz>
<satz n="Ts–213,45r_[2]"> das ist der <notation>a+b</notation> Satz XBBB nnn Don't forget me <lb/>this weekend </satz>
</ab>
</text>
</TEI>
```

Das **XSL-Stylesheet** in der Datei *mywitt.xsl* definiert die eigentliche Umw andlung:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
    <html>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
```

```
    <body>
    <xsl:apply-templates/>
    </body>
    </html>
</xsl:template>

<xsl:template match="ab">
    <p>
    <xsl:apply-templates/>
    </p>
</xsl:template>

<xsl:template match="satz">
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="notation">
    <code>
    <xsl:apply-templates/>
    </code>
</xsl:template>

<xsl:template match="alternative">
    <SPAN style="background-color:#FFFF00;">
    <xsl:apply-templates/>
    </SPAN>
</xsl:template>

<xsl:template match="lb[@rend='hyphen']">
    <xsl:value-of select="."/><xsl:text>-</xsl:text>
    <br/>
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="lb">
    <br/>
    <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

Damit das XML-Dokument das Stylesheet anwenden kann, muss es in der XML Datei eingebunden sein. Die **Einbindung** erfolgt im XML-Header über die Anweisung:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE TEI SYSTEM "file:/mounts/Users/cisintern/max/workspace/wittgenstein/texte/DTD/CISWAB.dtd">
<?xml-stylesheet type="text/xsl" href="mywitt.xsl"?>

<TEI>
<teiHeader>
<fileDesc>
<title>[DRAFT!] (2009–): Wittgenstein TS 213: Ts–213.
xml</title>
<author>Ludwig Wittgenstein</author>
<editor><persName>Alois Pichler</persName>
<orgName ref="http://wab.aksis.uib.no/">Wittgenstein
Archives at the University of Bergen (WAB)</
orgName>
</editor>
</fileDesc>
</teiHeader>
<text>
<ab n="Ts–213,44r">
<satz n="Ts–213,44r_[1]"> das ist der Satz AAA nnn </
satz>
<satz n="Ts–213,44r_[2]"> das ist der Satz BBB nnn <
pb facs="Ts–213,45r"/> Don't forget me <lb/>
this week–<lb/>end </satz>
</ab>
<ab n="Ts–213,45r">
<satz n="Ts–213,45r_[1]"> das ist der Satz XAAA nnn
```

```
</satz>
<satz n="Ts-213,45r_[2]"> das ist der <notation>a+b</
notation> Satz XBBB nnn Don't forget me <lb/>
this weekend </satz>
</ab>
</text>
</TEI>
```

7 Übungen

1. Welcher Unterschied wird zwischen Wohlgeformtheit und Gültigkeit bei XML-Dokumenten gemacht?
2. Wie sieht die DTD *CISWAB.dtd* zu folgendem XML-Dokument aus:

```
<TEI>
<teiHeader>
<fileDesc>
<title>[DRAFT!] (2009–): Wittgenstein TS 213: Ts-213.xml</title>
<author>Ludwig Wittgenstein</author>
<editor><persName>Alois Pichler</persName>
<orgName ref="http://wab.aksis.uib.no/">Wittgenstein Archives at the University of Bergen (WAB)</orgName>
</editor>
</fileDesc>
</teiHeader>
<text>
<ab n="Ts-213,44r">
<satz n="Ts-213,44r_[1]"> das ist der Satz AAA nnn </satz>
<satz n="Ts-213,44r_[2]"> das ist der Satz BBB nnn <pb facs="Ts-213,45r"/> Don't forget me <lb/>this week-<lb/>end </satz>
</ab>
<ab n="Ts-213,45r">
<satz n="Ts-213,45r_[1]"> das ist der Satz XAAA nnn
</satz>
<satz n="Ts-213,45r_[2]"> das ist der <notation>a+b</notation> Satz XBBB nnn Don't forget me <lb/>this weekend </satz>
</ab>
</text>
</TEI>
```

3. Ein Vorteil von XML-Schema ist, gültigen Inhalt von XML-Elementen genauer zu definieren. Oftmals haben sichere Passwörter eine mindest- wie auch höchst-Zahl von Stellenangaben. Schreibe die passende Schemadefinition des XML-Elementes , dessen Inhalt mindestens 5 und höchstens 10 Zahlen erwartet.

4. Mittels XSLT lassen sich XML-Dokumente automatisch transformieren. Hierfür wird u.a. die Schleifenfunktion bereitgestellt, die automatisch jedes Element von *select* durchsucht. Zu dem erweiterten Einführungsbeispiel (folgend) soll jetzt ein Stylesheet den XML-Inhalt in HTML als Tabelle darstellen.

```
<?xml version="1.0" encoding="UTF-8"?>
<buecher>
    <buch>
        <titel>Der Traum</titel>
        <autor />
    </buch>
    <buch>
        <titel>Stolz und Vorurteil</titel>
        <autor> Jane Austen </autor>
    </buch>
</buecher>
```

Autor und Titel sollen dabei in einer Tabelle aufgelistet werden wie folgt

der Titel	der Autor

Der Traum	
Stolz und Vorteil	Jane Austen

Vervollständige folgendes Stylesheet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/_______________">
        <html>
        <body>
        <table border="1">
        <tr>
        <th>___________</th>
        <th>__________</th>
        </tr>
        <xsl:for-each select="buch">
        <tr>
        <td><xsl:value-of select="________"/></td>
        <td><xsl:value-of select="______"/></td>
        </tr>
        </xsl:for-each>
        _________
        __________
        </html>
    </xsl:template>
</xsl:stylesheet>
```

8 Literaturverzeichnis

Helmut Vonhoegen, Einstieg in XML, Bonn 2011, Galileo Computing
Holger Meuss (2002) XML-Anfragesprachen
http://www.cis.uni-muenchen.de/kurse/max/korpunix/scripten/Meuss-Skriptum.pdf (Abgerufen 11.05.2014)