FHD

3

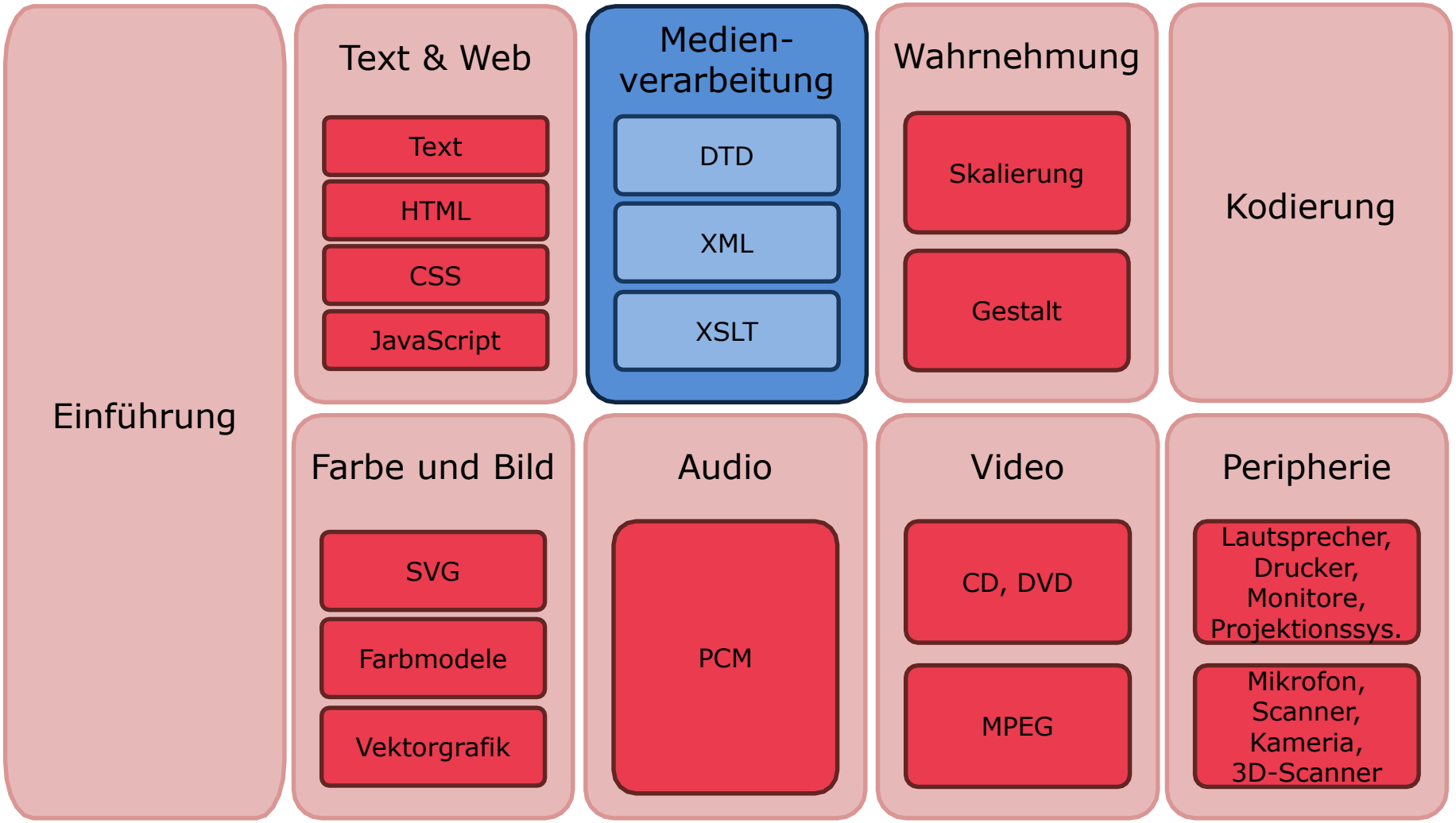
mit Material von G. Weber (TU Dresden):

Einführung in die Medieninformatik

Medienverarbeitung

Prof. Dr.-Ing. Tenshi Hara
fhd-emi@lern.es

VORLESUNGSÜBERSICHT



In a Nutshell: Document Type Definition (DTD)

Document Type Definition (DTD)

- Regelsatz zur Definition einer Klasse ähnlicher Dokumente
- besteht aus
  - möglichen Elementtypen
  - erlaubten Attributen von Elementen
  - Entitäten
  - Notationen
- legt fest
  - Reihenfolge
  - mögliche Schachtelung der Elemente
  - Art des Inhalts von Attributen

ELEMENTDEFINITION

- Definition ähnlich EBNF
- Kommentare mittels `<!--` und `-->`
- Inhaltsdefinition

  `#CDATA` – unverarbeitete Zeichenfolgen (Character Data)

  `#PCDATA` – verarbeitbare Zeicheninhalte (Parsed Character Data)

  `EMPTY` – kein Inhalt

  `ANY` – beliebiger Inhalt

- Struktur

  `,` – Reihenfolge (v.L.n.R.)

  `|` und `( )` – wie bei EBNF

- Wiederholungen/Anzahl

  `*` , `+` und `?` – wie bei EBNF

Beispieldefinition: HTML-Elemente

```
<!ELEMENT html (head, body)>

<!ELEMENT hr EMPTY>

<!ELEMENT div (#PCDATA | p | ul | ol | dl | table | pre | hr |
        h1|h2|h3|h4|h5|h6 | blockquote | address |
        fieldset)*>

<!ELEMENT dl (dt|dd)+>

<!-- u.s.w. -->
```

In a Nutshell: Extensible Markup Language (XML)

Extensible Markup Language (XML)

- Auszeichnungssprache zur Beschreibung hierarchischer Strukturen
- XML-Dokumente
  - bestehen aus UTF-8-Zeichen
  - *können* einer DTD unterliegen
  - sind in Elemente organisiert
  - sind wohlgeformt, wenn alle Regeln eingehalten werden
  - *können* auf eine Grammatik verweisen
  - gültig (valid), wenn *wohlgeformt und grammatisch korrekt*
  - Dateiname ist irrelevant

XML – Elemente

- wichtigstes Struktureinheit eines XML-Dokumentes;
  Träger der Information
- kann Text und weitere Elemente enthalten
- bildet Knoten des Strukturbaumes eines XML-Dokumentes
- Name ist in DTD-freien XML-Dokumenten frei wählbar, ansonsten muss dieser im DTD deklariert sein und das Element muss sich in einer zugelassenen Position innerhalb des Strukturbaumes gemäß DTD befinden.

XML – Wohlgeformtheit (1/3)

- alle XML-Regeln müssen einhalten werden
- XML-Dokumente können aus Elementen, Attributen und textuellen Inhalten bestehen
- Elemente werden durch ihren Elementnamen gekennzeichnet, sowie durch die Metazeichen `<`, `/` und `>`
- Elemente mit Inhalt besitzen einen Beginn- und einen Endauszeichner
  - Beginnauszeichner haben die Form `<elementname>`
  - Endauszeichner haben die Form `</elementname>`
  - Elemente ohne Inhalt können aus nur einem Auszeichner bestehen, der mit `/>` abschließt: `<elementname/>`

XML – Wohlgeformtheit (2/3)

- Elemente können Attribute haben, welche eine Verknüpfung mit Eigenschaften deklarieren
- Elemente dürfen nicht mehrere Attribute mit demselben Namen besitzen
- Attribute tragen Informationen und spezifizieren Elemente näher
  - gehören immer einem Element
  - besitzen einen Namen und einen von " begrenzten Wert
  - Attributsname und –wert werden mit = verknüpft:
    ```
    <elementname attributname="wert"/>
    ```
  - der Attributswert kann auch leer sein:
    ```
    <elementname attributname=""/>
    ```

XML – Wohlgeformtheit (3/3)

- XML-Dokumente unterliegen einigen simplen Syntaxregeln
  - XML-Dokumente haben immer einen Inhalt, mindestens die XML-Deklaration (`<?xml version="1.0" encoding="UTF-8"?>`) und ein Wurzelelement
  - es gibt genau ein Wurzelelement
  - Beginn- und Endauszeichner sind immer paarig und ebenentreu
  - Elemente können Elemente enthalten
  - Elemente können Textzeichen enthalten
  - Elementnamen sind case-sensitive (Groß-/Kleinschreibung wichtig!)
  - Anfang muss dem Ende entsprechen

XML – Grammatik

- einerseits durch Struktur der DTD definiert
- andererseits in XML Schema (XSD) möglich
  - ähnlich DTD, aber selbst XML-konform und mehr Datentypen unterstützend
  - beschreibt XML-Schema-Instanzen (einzelne Dateien) und Instanzengruppen (Typen von Dateien)
  - Basisdatentypen: xs:string, xs:decimal, xs:integer, xs:float, xs:boolean, xs:date, xs:time
  - Listen und Unions (bestehend aus atomaren Elementen und Listen)

XSD-Beispiel

```
<xs:complexType name="student">
   <xs:sequence>
      <xs:element name="nachname" type="xs:string"/>
      <xs:element name="vorname" type="xs:string"/>
      <xs:element name="geburtstag" type="xs:date"/>
      <xs:element name="semester" type="xs:integer"
         minInclusive="1" maxInclusive="20"/>
      <xs:element name="studiengang" type="xs:integer"
         minInclusive="1"/>
      <xs:element name="kommentar" type="xs:string"
         minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="matrikel" type="xs:integer"/>
</xs:complexType>
```

XML-Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:studenten="file://D:/Verwaltung/studentenschema.xsd">
<studenten>
   <student matrikel="1">
      <nachname>Mustermann</nachname>
      <vorname>Peter</vorname>
      <geburtstag>1996-07-18</geburtstag>
      <semester>2</semester>
      <studiengang>4</studiengang>
   </student>
   <student matrikel="7">
      <nachname>Doe</nachname>
      <vorname>Jane</vorname>
      <geburtstag>1998-02-21</geburtstag>
      <semester>2</semester>
      <studiengang>3</studiengang>
      <kommentar>Austauschstudentin</kommentar>
      <kommentar>keine Mensa-Berechtigung</kommentar>
   </student>
   <!-- u.s.w. -->
</studenten>
```

Medien und Inhalte mit XML

Medienverarbeitung mit XML

HTML und Javascript sind Beispiele für formale Sprachen, die Medien beschreiben und automatisch verarbeitet (bzw. geprüft) werden können.

→ XML ist eine Metasprache, um solche Sprachen und ihre Interaktion zu formalisieren.

Beispiele:

- HTML beschreibt Text
- SMIL dient zur Synchronisation von Audio, Video mit Text, Bildern
- MusicXML beschreibt Notensatz
- MathML erlaubt den Austausch mathematischer Formeln
- QTI legt Prüfungsfragetechniken fest
- SVG beschreibt Graphiken
- Browser unterstützen Webseiten mit HTML-, SVG- und MathML-Inhalten

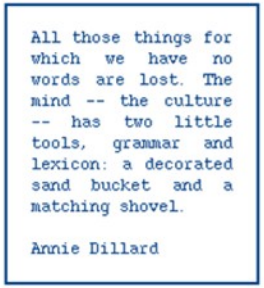
XML UND SGML

- XML basiert auf ISO 8879: Standard Generalized Markup Language (SGML) for Information Processing, Text and Office Systems
- XML kann für Langzeitarchivierung geeignet sein, wenn die bedeutungstragenden Elemente der Medien erhalten bleiben (auch, wenn sich Darstellungstechniken verändern)

Wo wird SGML verwendet?

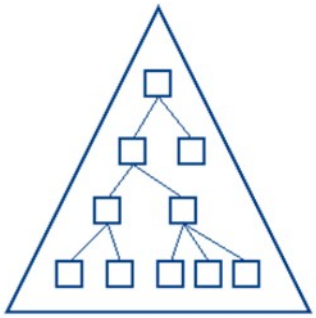
- Euro-Tunnel: 7 GB Daten in SGML
- Airbus: technische Dokumentation von Flugzeugen
- US-DoD verlangt(e) SGML für alle Produktdokumentationen
- Zeitungsverlage verwenden SGML zur Archivierung

Medien in Dokumenten



Inhalt

Semantik



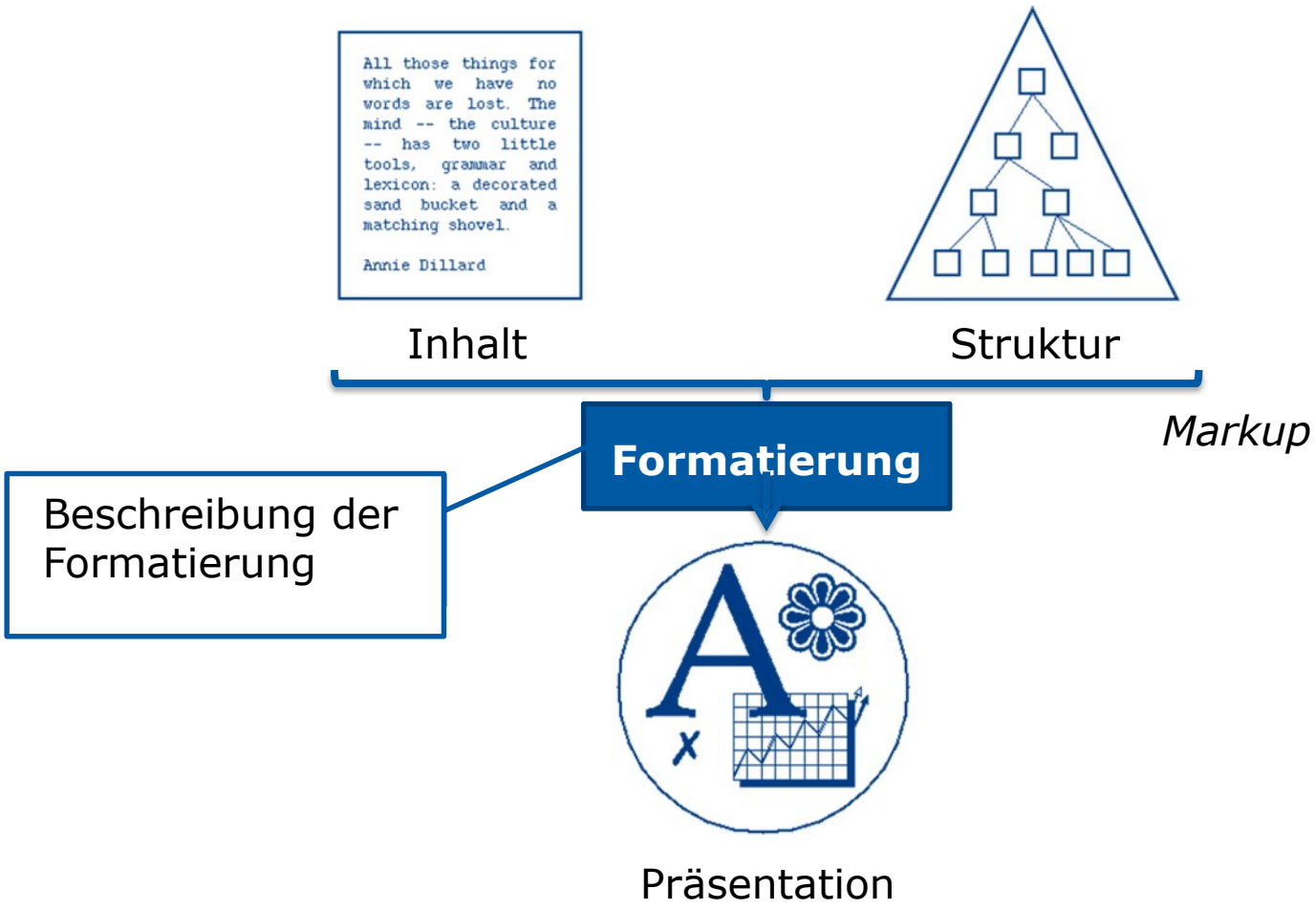
Struktur

Syntax

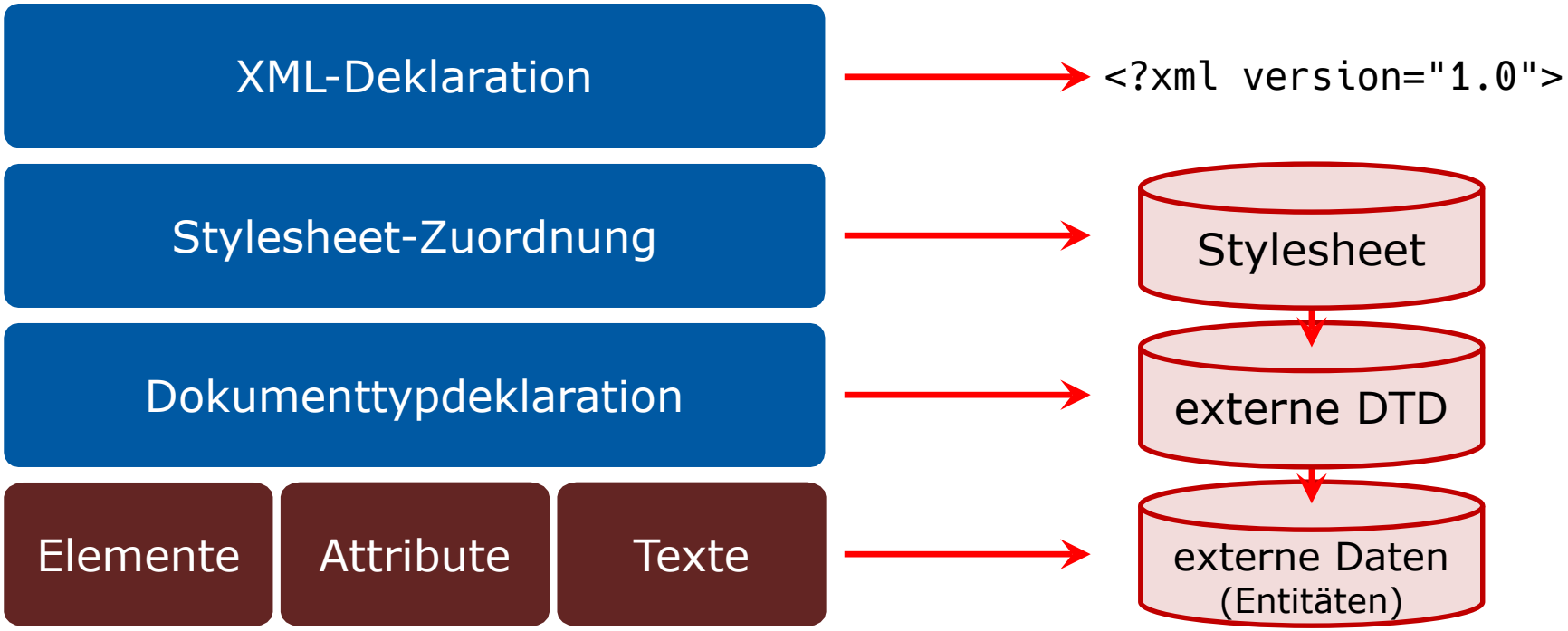
Präsentation

Lexikalische Information

Dokumente mit XML

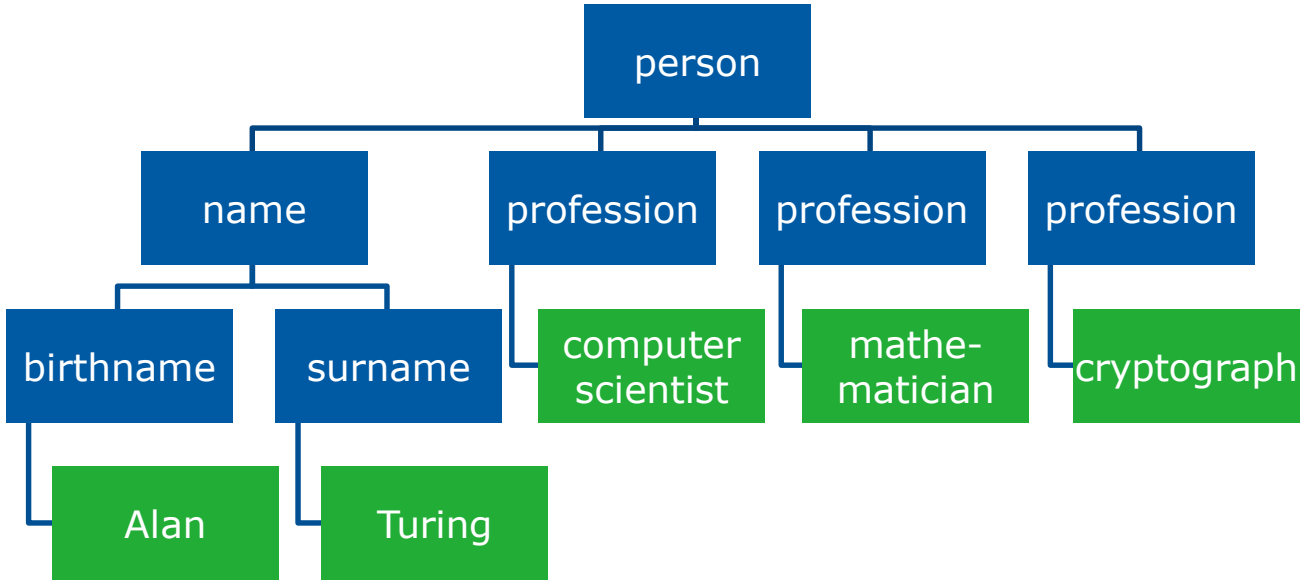


Vollständiger Aufbau eines XML-Dokuments



XML-BÄUME

```
<person>
   <name>
      <birthname>Alan</birthname>
      <surname>Turing</surame>
   </name>
   <profession>computer scientist</profession>
   <profession>mathematician</profession>
   <profession>cryptograph</profession>
</person>
```



Syntax:
hierarchischer Aufbau, bestehend aus Elementen und Textzeichen

Begriffe:

- Element
- Knoten (Tag)
- Vorgänger
- Nachfolger
- Nachbar
- Wurzel

GEMISCHTER INHALT

Textzeichen und Elemente werden als Nachbarn behandelt oder verschachtelt dargestellt.

Beispiel:

```
<biography><name><first_name>Alan</first_name>
<last_name>Turing</last_name></name> war einer der Ersten die
man als <emphasize>Informatiker</emphasize> bezeichnen kann.
W&auml;hrend er eine Vielzahl von Beitr&auml;gen geliefert
hat, ist er vor allem f&uuml;r den <emphasize>Turing
Test</emphasize> und die <emphasize>Turing
Maschine</emphasize> bekannt.</biography>
```

Schnellübung: Zeichnen Sie den XML-Baum!

Elemente tragen Attribute

- ein Attribut ist ein Paar `Attributname="Attributwert"`, das nach dem Anfang eines Elements an den Elementnamen angehängt wird
- Attributwerte werden mit `'` oder `"` begrenzt
- ein Element kann mehrere Attribute haben
- es muss zwischen Element beschreibenden Attributen und Eigenschaften von Elemententitäten unterschieden werden:

```
<person dob="1912/06/23" dod="1954/06/07">
   Alan Turing
</person>
```

gegenüber

```
<person>
   <dob>1912/06/23</dob>
   <dod>1954/06/07</dod>
   <name>Alan Turing</name>
</person>
```

Elementnamen sind XML Namen

- erstes Zeichen muss ein Buchstabe sein; nach vereinbarter Kodierung also: a, Ö, Ω, ç, _ (Unterstrich), - (Bindestrich), . (Punkt)
- ab zweitem Zeichen danach auch Ziffern zugelassen
- Leerzeichen, Doppelpunkt, Schrägstrich, … sind reservierte Metazeichen

Beispiele:

- wohlgeformt
  - `<Drivers_License_Number>`
  - `<month-day-year>29/3/2002</month-day-year>`
  - `<_4-lane>I-610</_4-lane>`
- nicht wohlgeformt
  - `<Driver's_licence_number>`
  - `<month/day/year>`
  - `<first name>`
  - `<4-lane>`

Namen für Entitäten

- Entitäten sind Platzhalter für andere Texte bzw. anderen Markup
- Entitäten kodieren Metazeichen:

  < `<`

  & `&`

  > `>`

  " `&quot;`

  ' `&apos;`

→ Vermeiden von Fehlinterpretationen

Beispiel:

```
<image source='oreilly.gif' width='122' height='66'
   alt='erh&auml;tlich bei O&apos;Reilly Books'>
```

CDATA-Abschnitte

- CDATA-Abschnitte werden für beliebige Zeichenketten benutzt
- sie werden durch die Metazeichen `!`, `[`, und `]` gekennzeichnet
- innerhalb eines CDATA Abschnitts sind alle anderen Metazeichen zulässig:

```
<![CDATA[
   <svg xmlns="http://www.w3.org/2000/svg"
      width="12cm" height="10cm">
      <elipse rx="110" ry="130" cx="1cm" cy="1cm"/>
      <rect x="4cm" y="1cm" width="3cm" height="6cm"/>
   </svg>
]]>
```

Achtung: `]]` kann nicht innerhalb des CDATA-Abschnitts verwendet werden, diese Zeichen wären sonst mehrdeutig. `]]` markiert immer das Ende von CDATA.

Weitere Regeln des Mark-up

- Kommentare können überall eingefügt werden (solange die Namen erkennbar bleiben) und sind v.d.F. `<!-- ein Kommentar -->`
- Verarbeitungsanweisungen (processing instructions) weisen auf Verarbeitungsschritte hin

```
<?robots index="yes" follow="no"?>
<?xml-stylesheet href="person.css" type="text/css"?>
```

- XML-Deklaration
  - steht am Anfang einer XML Datei, ist kein Element
  - Kodierung nach UTF-8 ist voreingestellt
  - `standalone="no"` kündigt an, dass die Syntaxbeschreibung (z.B. DTD) beigefügt ist
  - Beispiel:

```
<?xml version="1.0" encoding="US-ASCII" standalone="yes"?>
```

Validieren von XML-Dokumenten

- Wohlgeformtheit ist zur Vermeidung von Fehlinterpretationen notwendig
- Prüfung mittels
  - Websites, z.B.:
    - https://www.xmlvalidation.com
    - www.cogsci.ed.ac.uk/%7Erichard/xml-check.html
  - mittels Programmen wie Xerces: `java sax.SAXCount -v doc.xml`
- nur wohlgeformte XML-Dateien können automatisch verarbeitet werden
- wenn z.B. Browser Ausnahmen zulassen, und die Dokumente darauf bauen, dann werden XML Dokumente unterschiedlich behandelt und dargestellt

Definition von Inhaltstypen mit DTD

Syntaxbeschreibungen

- Sprachen wie HTML und SVG basieren auf Syntaxbeschreibungen, um die Art der Elemente und deren Attribute festzulegen
- XML kennt zwei Arten von Syntaxbeschreibungen:
  - Document Type Definition (DTD)
    - definieren die Struktur von XML Dokumenten durch Deklaration von Regeln
    - sind selbst kein XML
    - geben nur die Regeln an, aber nicht die eigentlichen Dokumente
    - geben keine Textzeichen vor
    - beschreiben nicht die Semantik der Elemente bzw. der Daten
  - XML Schema Definition (XSD)

EINFACHES DTD-BEISPIEL

```
<!ELEMENT person (name, profession*)>
<!ELEMENT name (first_name, last_name)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT last_name (#PCDATA)>
<!ELEMENT profession (#PCDATA)>
```

DTD kann vorliegen

- zusammen mit dem XML Dokument in derselben Datei
- an anderer Stelle im Internet
- in einer anderen Datei
- gar nicht (d.h. das verarbeitende Programm trifft implizite Annahme)

Beispiel:

```
<!DOCTYPE person SYSTEM "https://www.ibiblio.org/xml/books/
   biblegold/source/34/person.dtd">
```

Interne DTDs

```
<?xml version="1.0"?>
<!DOCTYPE person [
   <!ELEMENT person (name, profession*)>
   <!ELEMENT name (first_name, last_name)>
   <!ELEMENT first_name (#PCDATA)>
   <!ELEMENT last_name (#PCDATA)>
   <!ELEMENT profession (#PCDATA)>
]>
<person>
   <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
   </name>
   <profession>computer scientist</profession>
   <profession>mathematician</profession>
   <profession>cryptographer</profession>
</person>
```

Deklaration von Elementen (1/2)

Element besteht

- nur aus Textzeichen einschließlich Entitäten: `#PCDATA`
- einem einzelnen Element: `<!ELEMENT fax (phone_number)>`
- einer Sequenz von Elementen:
  - `<!ELEMENT name (first_name+, last_name)>`
  - in dieser Reihenfolge, ohne Auslassungen, ohne Erweiterung
  - Anzahl der Nachfolger muss durch ein Suffix festgelegt werden
    - `?` kein oder ein Element (d.h. Element ist optional)
    - `*` kein Element oder eine beliebige Anzahl von Elementen
    - `+` mindestens ein Element (oder mehrere)

Deklaration von Elementen (2/2)

- Alternativen fassen Regeln zusammen:

```
<!ELEMENT transport (car | train | plane)>
```

- Klammern fassen Sequenzen zusammen
  - unterstützen Verschachtelung
  - tragen evtl. auch ein Suffix `?`, `*`, `+`
  - bilden Alternativen: `<!ELEMENT point ( (a,b) | (y,z) )>`

Schnellübungen: Überlegen Sie DTDs für

- Ein Punkt soll entweder durch kartesische Koordinaten (x,y) oder Polarkoordinaten (ρ,ϴ) beschrieben werden
- Ein Polygon besteht aus drei oder mehr Punkten mit kartesischen oder polaren Koordinaten
- Der Name einer Person besteht genau aus einem Vornamen, einer beliebigen Anzahl von weiteren Vornamen, sowie einem Nachnamen

Besonderheiten der Deklaration

- Textzeichen mit Elementen mischen:

```
<!ELEMENT paragraph (#PCDATA | name | profession
   |footnote | emphasize | date)*>
```

- leere Elemente enthalten keine Nachfolger, auch keine Textzeichen

```
<!ELEMENT br EMPTY>
```

  - Vorsicht: muss beendet werden, oder ist per Abkürzung zu schreiben, aber es gibt Sonderfälle:
    - `<br/>` → XML
    - `<br />` → XHTML

- any

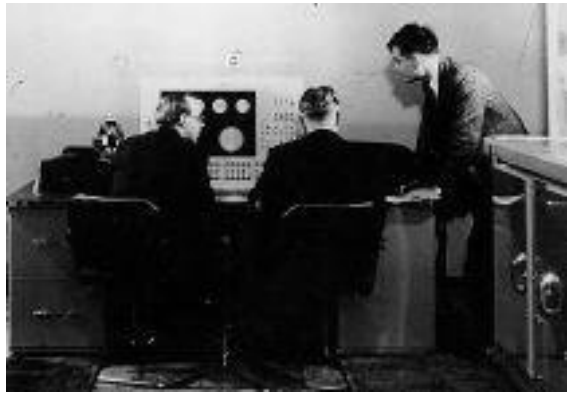
```
<!ELEMENT page ANY>
```

  - ist wenig aussagekräftig, aber kann zu Beginn des Entwurfs von Regeln hilfreich sein
  - würde es erlauben, unbekannte XML Dokumente zu integrieren

Deklaration von Attributen

für jedes Element können Attribute festgelegt werden:

```
<!ATTLIST image source CDATA #REQUIRED
                 width CDATA #REQUIRED
                height CDATA #REQUIRED
                   alt CDATA #REQUIRED
>
```



Beispiel (Instanz):

```
<image source="bus.jpg"
        width="122"
       height="345"
          alt="Alan Turing am Manchester Computer"/>
```

VERSCHIEDENE ATTRIBUTTYPEN

Um den Typ von Attributwerten festzulegen, werden bei der Deklaration folgende Schlüsselwörter verwendet:

- `CDATA` jegliche Zeichenkette, d.h. Text, URL, Preise, E-Mail-Adressen, …
- `NMTOKEN` ein wohlgeformter XML-Name
- `NMTOKENS` ein oder mehrere XML-Namen (durch Leerzeichen getrennt)

  Beispiel zur Aufzählung von Attributwerten:

```
<!ATTLIST date month (Jan | Feb | Mar | Apr | May | Jun |
   Jul | Aug | Sep | Oct | Nov | Dec) #REQUIRED>
```

- `ENTITY` enthält den Namen einer Entität (definiert an anderer DTD-Stelle)
- `ENTITIES` mehrere Entitäten (durch Leerzeichen getrennt)
- `ID` ein eindeutiger XML-Name (nicht an anderer Stelle als ID benutzt)
- `IDREF` reiner XML-Name der sich auf eine ID dieses Dokuments bezieht
- `IDREFS` Sequenz von Ids (durch Leerzeichen getrennt)

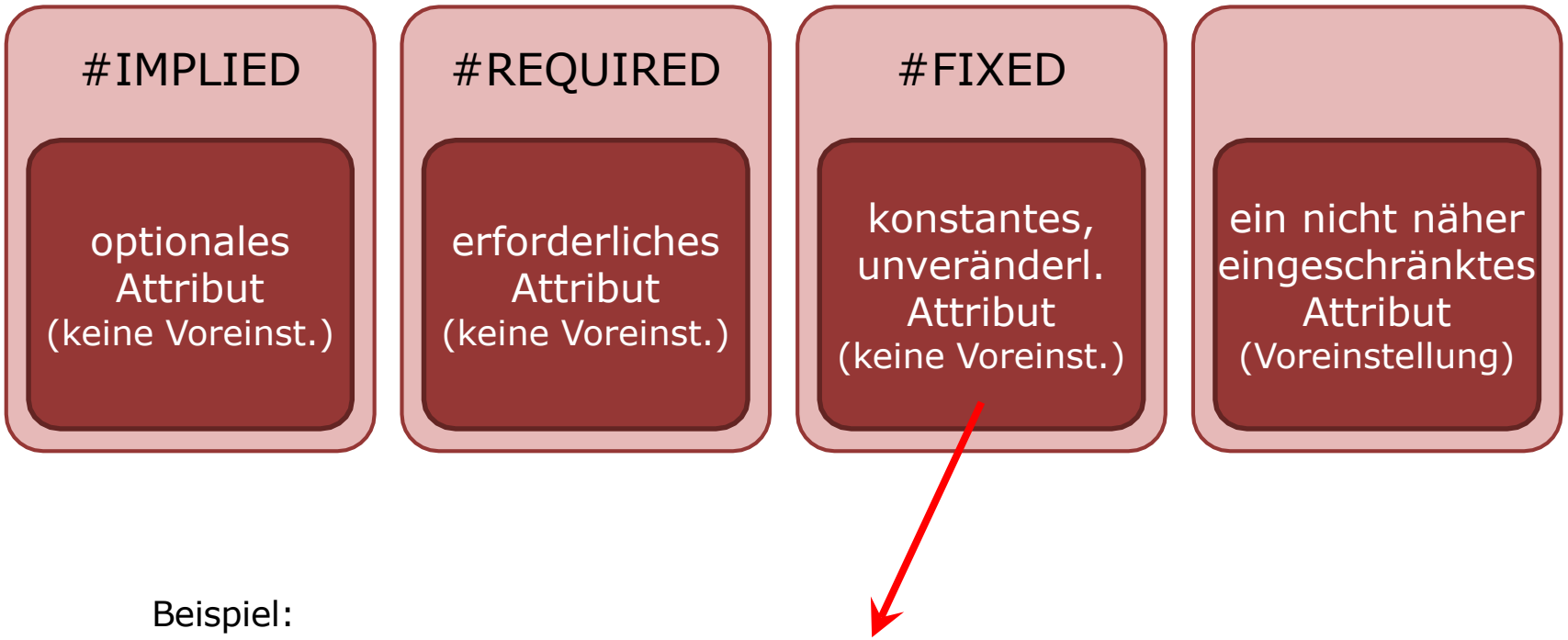
Beispiel für IDREFS Attribut

```
<!ATTLIST project project_id ID #REQUIRED
          team IDREFS #REQUIRED>
<!ATTLIST person ssn ID #REQUIRED
          assigments IDREFS #REQUIRED>

<project project_id="p1" team="ss123-45-6789">
   <goal>Entwicklung eines strategischen Plans</goal>
</project>
<project project_id="p2" team="ss123-45-6789 ss8765-43-210">
   <goal>Einführung von Linux</goal>
</project>

<person ssn="ss123-45-6789" assignments="p1 p2">
   <name>Max Mustermann</name>
</person>
<person ssn="ss8765-43-210" assignments="p2">
   <name>Petra Beispielfrau</name>
</person>
```

Voreinstellungen der Attribute



Beispiel:

```
<!ATTLIST biography xmlns:xlink CDATA #FIXED "http://www.w3.org/xlink">
```

→ angegebene URL ist immer für das Element biography festgelegt

Beispiel: Personendaten

```
<?xml version="1.0"?>
<!DOCTYPE person [
   <!ELEMENT person (name+, profession*)>
   <!ELEMENT name EMPTY>
   <!ATTLIST name first CDATA #REQUIRED last CDATA #REQUIRED>
   <!-- die Attribute first bzw. last sind notwendig, können
   jedoch leer bleiben, z.B. <name first="Bobo" last=""/> -->
   <!ELEMENT profession EMPTY>
   <!ATTLIST profession value CDATA #REQUIRED>
]>
<person>
   <name first="Alan" last="Turing"/>
   <profession value="computer scientist"/>
   <profession value="mathematician"/>
   <profession value="cryptograph"/>
</person>
```

Beispiel: Textverarbeitung (DTD-Teil)

```
<!DOCTYPE biography [
   <!ELEMENT biography ( #PCDATA | image | paragraph
      | definition | graph | person | profession | emphasize
      | birthname | surname | footnote | date )* >
   <!ELEMENT footnote (#PCDATA)>
   <!ATTLIST footnote source CDATA #REQUIRED>
   <!ELEMENT person (birthname, surname)>
   <!ATTLIST person dob CDATA #REQUIRED dod CDATA #REQUIRED>
   <!ELEMENT birthname (#PCDATA)>
   <!ELEMENT surname (#PCDATA)>
   <!ELEMENT image EMPTY>
   <!ATTLIST image source CDATA #REQUIRED alt CDATA #IMPLIED
      width NMTOKEN #REQUIRED height NMTOKEN #REQUIRED>
   <!ELEMENT date (day, month, year)>
   <!ELEMENT day (#PCDATA)>
   <!ELEMENT month (#PCDATA)>
   <!ELEMENT year (#PCDATA)>
   <!ELEMENT profession (#PCDATA)>
   <!ELEMENT emphasize (#PCDATA)>
   <!ATTLIST emphasize xlink:href CDATA #IMPLIED>
   <!ATTLIST emphasize xlink:type CDATA #IMPLIED>
]>
```

Beispiel: Textverarbeitung (XML-Teil)

```
<?xml version="1.0"?>
<biography><image source="http://www.turing.org.uk/turing/pi1/bus.jpg"
width="152" height="345"/><person dob="1912/06/23" dod="1954/06/07">
<birthname>Alan</birthname><surname>Turing</surname></person> war
einer ersten die den Namen <emphasize>Informatiker</emphasize> tragen
durften. Obwohl er unz&auml;hlige Beitr&auml;ge zur Informatik
geleistet hat, ist er am besten f&uuml;r den <emphasize>Turing
Test</emphasize> und die <emphasize>Turing Maschine </emphasize>
bekannt. <surname>Turing</surname> war auch ein ausgewiesener
<profession>Mathematiker</profession> und <profession>Kryptograph
</profession>. Seine Arbeit war ma&szlig;gebend um den Kode der Enigma
Maschine zu entschl&uuml;sseln<footnote source="The Ultra Secret, F.W.
Winterbotham, 1974">1</footnote>. Er nahm sich das Leben am
<date><day>7</day><month>Juni</month><year>1954</year></date> nachdem
er der Homosexualit&auml;t f&uuml;r schuldig befunden wurde und
gezwungen wurde weibliche Hormone einzunehmen<footnote source="Alan
Turing: the Enigma, Andrew Hodges, 1983">2</footnote>.</biography>
```

Namensräume (namespaces)

- XML-Namen gehören Namensräumen (Syntaxbeschreibung) an
- Präfixe unterscheiden identische Elemente verschiedener Namensräume
  → Elemente verschiedener DTDs müssen unterschieden werden
  - Beispiel: `svg:img` ist verschieden von `html:img`
  - Namensräume werden bei Elementen durch das allgemein gültige Attribut `xmlns` deklariert:

```
<svg xmlns="http://www.w3.org/2000/svg"
   width="12cm" height="10cm">
   <ellipse rx="110" ry="130"/>
</svg>
<html xmlns:svg="http://www.w3.org/2000/svg">
   <svg:ellipse rx="110" ry="130"/>
</html>
```

  - Die Bindung gilt für darin verschachtelte Elemente

Beispiel mit drei Namensräumen

```
<?xml version="1.0"?>
<html
   xmlns="http://www.w3.org/1999/xhtml"
   xmlns:xlink="http://www.w3.org/1999/xlink">
<head><title>Drei Namespaces</title></head>
<body>
   <h1 align="center">Eine Ellipse und ein Rechteck</h1>
   <svg xmlns="http://www.w3.org/2000/svg"
      width="12cm" height="10cm">
      <ellipse rx="110" ry="130"/>
      <rect x="4cm" y="1cm" width="3cm" height="6cm"/>
   </svg>
   <p xlink:type="simple" xlink:href="ellipses.html">
      mehr zu Ellipsen
   </p>
   <p xlink:type="simple" xlink:href="rectangles.html">
      mehr zu Rechtecken
   </p>
   <hr />
</body>
</html>
```

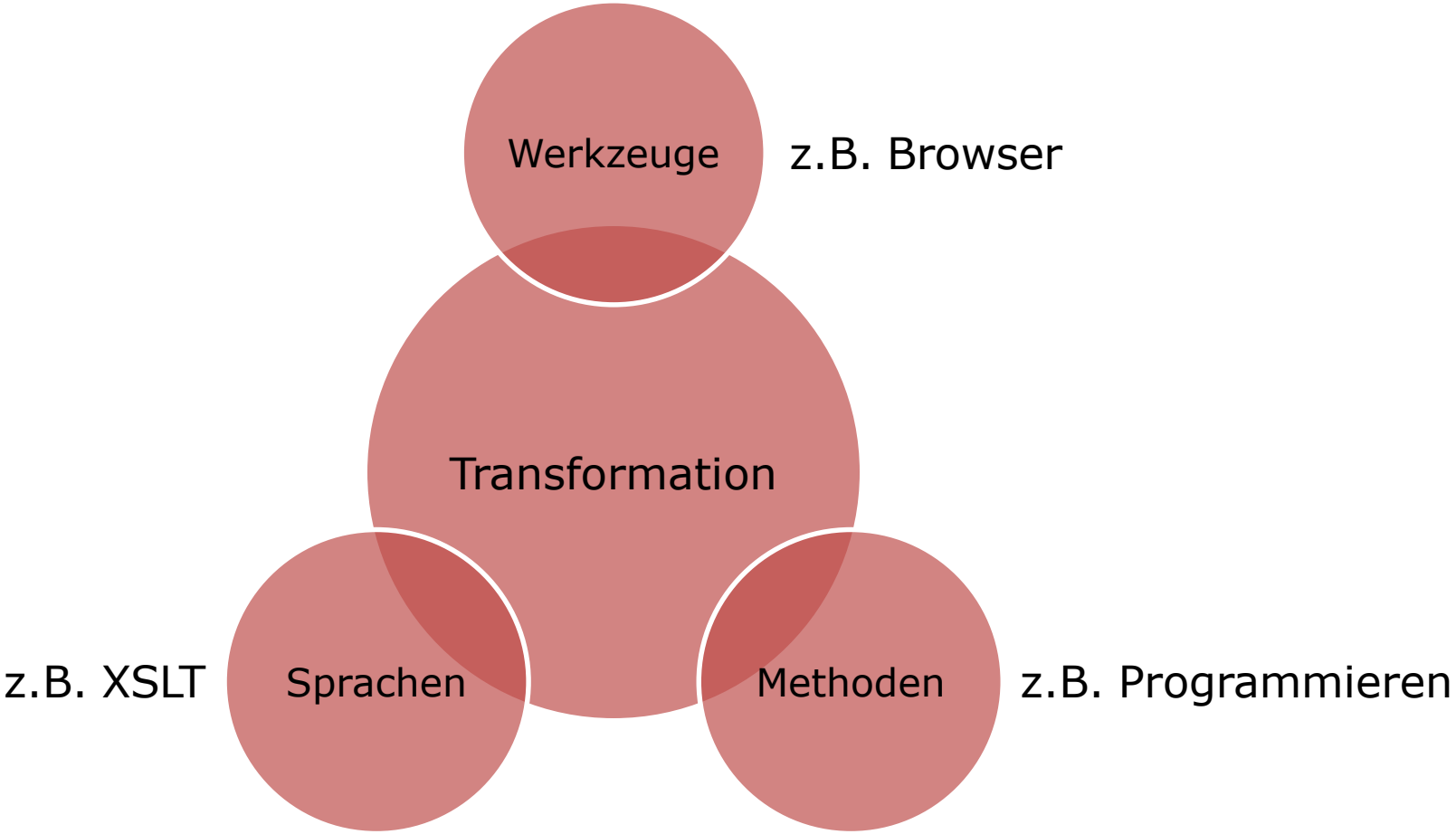
Grenzen der DTDs

- keine Datentypen für Elemente und Attribute; nicht möglich, folgendes zu unterscheiden:
  - Postleitzahlen sind CharData
  - Postleitzahlen haben 5 Ziffern
  - Postleitzahlen können führende 0en haben
  - Postleitzahlen besitzen kein Vorzeichen
- keine Modularisierung
- XML Schema sind selbst XML Dokumente, DTDs nicht

```
<xsd:schema xmlns:xsd=
"http://www.w3.org/1999/XMLSchema">
   <xsd:annotation>
      <xsd:documentation>
         schema for a book
      </xsd:documentation>
   </xsd:anotation>
   <xsd:element name="book"
      type="bookT" />
   <xsd:complexType name="bookT">
      <xsd:element name="title"
         type="xsd:string" />
      <xsd:element
         name="information"
         type="informationT" />
   </xsd:complexType>
</xsd:schema>
```

Extensible Stylesheet Language (XSL) und XSL Transformation (XSLT)

Konzepte der Informatik



XSL und XSLT

- XML ist geeignet, XML-Dokumente in XML zu verarbeiten
- Spezifikation dazu: http://www.w3.org/TR/xslt

Ziele

- Daten zwischen Anwendungen austauschen
- Daten von der Präsentation trennen
- Teilweise wird in mehreren Schritten vorgegangen
  - erste Verarbeitung: Transformation der Struktur
  - zweite Verarbeitung: Formatierung der Ausgabe (HTML, PDF, …)

„Hallo Welt!“ in XSLT – XML-Quelle und HTML-Ausgabe

helloworld.xml:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="helloworld.xsl"?>
<greeting>Hallo Welt!</greeting>
```

Ausgabe:

```
<html>
<head>
   <title>Wort zum Tag</title>
</head>
<body>
   <p>Hallo Welt!</p>
</body>
</html>
```

„Hallo Welt!“ in XSLT – XSL-Deklaration

helloworld.xsl verwendet 2 XML Sprachen: XSL und HTML

```
<?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>
         <head>
            <title>Wort zum Tag</title>
         </head>
         <body>
            <p><xsl:value-of select="greeting" /></p>
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

Merkmale von XSL

Verwenden von XML-Syntax

keine Seiteneffekte, alles muss deklariert werden,
Sequenz muss nicht immer eingehalten werden

Verarbeitung basiert auf Templates

XSLT 1.1

- mehrfache Ausgabedokumente
- mehrere Bäume
- Unterstützung von Java und EcmaScript

XSLT 2.0

- verwendet XSchema

Transformationen

Templates anwenden lassen

- `<xsl:apply-templates>`
- bearbeitet Elemente durch Anwendung der entsprechenden *Templateregeln*

Textzeichen („Wert“) aus einem Element holen

- `<xsl:value-of>`
- extrahiert *alle Informationen des Elements*

Templates wiederholt auf alle identischen Elemente anwenden

- `<xsl:for-each>`
- *jedes Element* wird verarbeitet

Vorlagenregeln (Template Rules)

match-Attribut

- benennt (addressiert) ein Element (mittels der XPath Sprache)
- Ausdruck `/` adressiert Wurzel
- Ausdruck `title` adressiert ein Element
- Ausdruck `chapter/title` adressiert Element innerhalb eines anderen

Weitere Regelelemente

- `xsl:message` gibt eine Nachricht beim transformieren aus
- `xsl:comment` kommentiert eine Regel
- `xsl:text (xml:space)` erzwingt Texte und Leerzeichen
- andere Elemente oder Textzeichen

Verschachtelte Templates

```
<xsl:template match="/">
   <xsl:message>Started!</xsl:message>
   <xsl:comment>generated from XSLT</xsl:comment>
   <html>
      <head>
         <title>Die erste Seite</title>
      </head>
      <body>
         <xsl:apply-templates/>
      </body>
   </html>
</xsl:template>
```

Selektiere alle Nachfolger des aktuellen Knotens im Dokumentenbaum, und wende die entsprechende Templateregel an.

Push-Verarbeitung: XSL-Definition

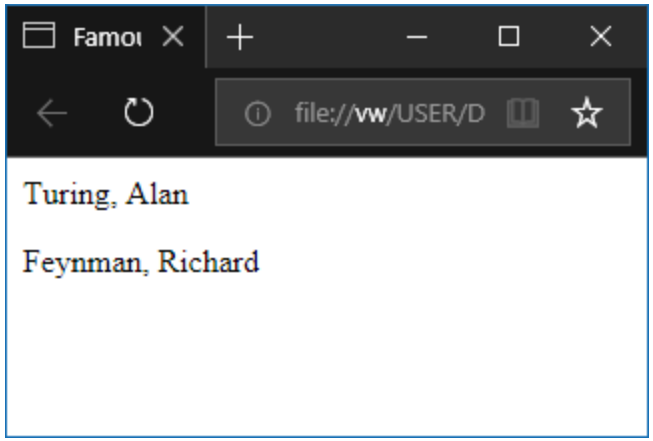
```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="people">
      <html>
         <head>
            <title>Famous Scientists</title>
         </head>
         <body>
            <xsl:apply-templates/>
         </body>
      </html>
   </xsl:template>
   <xsl:template match="name">
      <p>
         <xsl:value-of select="birthname"/>,
         <xsl:value-of select="surname"/>
      </p>
   </xsl:template>
   <xsl:template match="person">
      <xsl:apply-templates select="name"/>
   </xsl:template>
</xsl:stylesheet>
```

Push-Verarbeitung: XML-Dokument

```
<?xml version="1.0"?>
<people>
   <person born="1912" died="1954">
      <name>
         <birthname>Alan</birthname>
         <surname>Turing</surname>
      </name>
      <profession>computer scientist</profession>
      <profession>mathematician</profession>
      <profession>cryptographer</profession>
   </person>
   <person born="1918" died="1988">
      <name>
         <birthname>Richard</birthname>
         <initial>M</initial>
         <surname>Feynman</surname>
      </name>
      <profession>physicist</profession>
      <hobby>Playing the bongoes</hobby>
   </person>
</people>
```

Push-Verarbeitung: Ausgabe

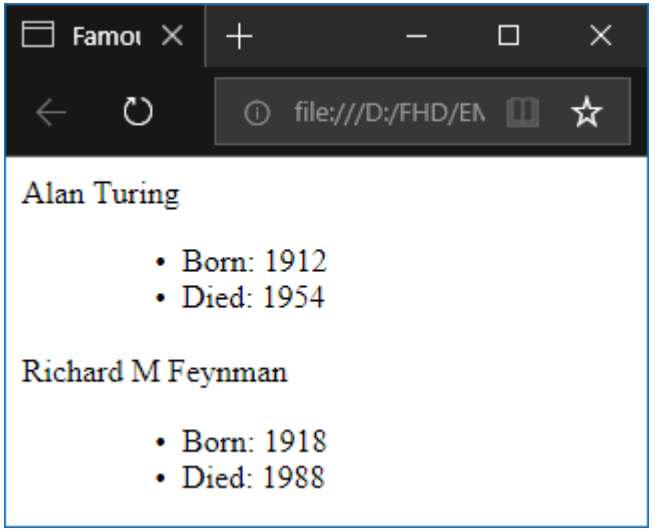
```
<html>
   <head>
      <title>Famous Scientists</title>
   </head>
   <body>
      <p>Turing, Alan</p>
      <p>Feynman, Richard</p>
   </body>
</html>
```



📖 **Rezept:**
Vorlagenregel für jedes Element, Ausgabe festlegen und Vorlagen steuern

Attribute lesen

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="people">
      <html>
         <head>
            <title>Famous Scientists</title>
         </head>
         <body>
            <dl><xsl:apply-templates /></dl>
         </body>
      </html>
   </xsl:template>
   <xsl:template match="person">
      <dt><xsl:apply-templates select="name" /></dt>
      <dd>
         <ul>
            <li>Born: <xsl:apply-templates select="@born" /></li>
            <li>Died: <xsl:apply-templates select="@died"/></li>
         </ul>
      </dd>
      <br/>
   </xsl:template>
</xsl:stylesheet>
```



Schnellübung:

Wie müsste die zugehörige `personattrib.xml` aussehen?

Pull-Verarbeitung

Für unregelmäßige/unvollständige Strukturen werdenVorlagenregeln für jeden Anwendungsfall geschrieben

- im Beispiel: falls kein zweiter Vorname existiert
- Verarbeitung nicht aller aber einiger gezielt ausgewählter Nachfolger eines Elements: `<xsl:apply-templates select="name">`

Beispiele für Anwendungsfälle bei Textverarbeitung sind

- Inhaltsverzeichnis
- Inhalt

Das Attribut `mode` erlaubt Anwendungsfälle zu unterscheiden und ist für `xsl:template` sowie `xsl:apply-templates` verwendbar

Falls kein Modus zutrifft, wird die Regel ohne Modusvorgabe verwendet

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

Stylesheet mit mehreren Modi

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="people">
        <html>
            <head><title>Famous Scientists</title></head>
            <body>
                <ul><xsl:apply-templates select="person" mode="toc"/></ul>
                <xsl:apply-templates select="person"/>
            </body>
        </html>
    </xsl:template>
    <!-- Table of Contents Mode Templates -->
    <xsl:template match="person" mode="toc">
        <xsl:apply-templates select="name" mode="toc"/>
    </xsl:template>
    <xsl:template match="name" mode="toc">
        <li><xsl:value-of select="surname"/>, <xsl:value-of select="birthname"/></li>
    </xsl:template>
    <!-- Normal Mode Templates -->
    <xsl:template match="person">
        <p><xsl:apply-templates/></p>
    </xsl:template>
</xsl:stylesheet>
```

XSLT und Namensräume

Wenn Namensräume in der Ausgabe benötigt werden, ist die Definition einer Namespace-ID notwendig:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:pe="http://namespaces.oreilly.com/people">
   <xsl:template match="pe:people">
      <html>
         <head>
            <title>Famous Scientists</title>
         </head>
         <body>
            <!-- … -->
```

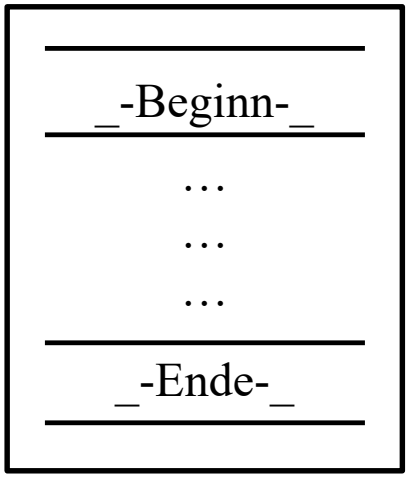
VARIABLEN

- Zuweisung: `<xsl:variable name="width" select="50"/>`
- globale Gültigkeit
- diverse Datentypen möglich
  - `string` jegliche Unicode Zeichen
  - `number` analog zu Java float
  - `boolean` 0..1 oder `false`...`true`
  - Element-Menge (set of nodes in source tree)
  - externe Objekte (z.B. Java Objekt, ...)
- Datentypen werden konvertiert
- Ausdrücke entstehen meist beim Adressieren eines Elements (siehe XPATH)

KONTROLLSTRUKTUR: <IF>

Bedingungen im Attribut match und mittels <if>

```
<xsl:template match="para">
   <xsl:if test="position()=1"></xsl:if>
      <hr/>_-Beginn-_<hr/>
   <xsl:apply-templates/>
   <xsl:if test="last()=1">
      <hr/>_-Ende-_<hr/>
   </xsl:if>
</xsl:template>
```



KONTROLLSTRUKTUREN: <FOR-EACH>

- Wiederholung für alle Elemente desselben Namens
- Beispiel:

```
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xsl:version="1.0">
   <head><title> a list of books </title></head>
   <body>
      <table>
         <xsl:for-each select="//book">
            <xsl:sort select="author"/>
            <tr>
               <td><xsl:value-of select="author"/></td>
               <td><xsl:value-of select="@category"/></td>
               <td><xsl:value-of select="price"/></td>
            </tr>
         </xsl:for-each>
      </table>
   </body>
</html>
```

Adressierung mit XPath

- einzelne Ortspfade
  - Wurzel: `/`
  - Nachfolger: `birthname`
  - Attributlokation: `@born`
  - weitere Knotenarten nach Funktion:
    - `comment()`
    - `text()`
    - `processing-instruction()`
- jegliches Element: `*`
- jegliches Attribut: `@*`
- auch mit Namespace-Präfix möglich:
  - `svg:*`
  - `@xlink:*`

ZUSAMMENGESETZTE ORTSPFADE

- Verbinden einzelner Ortspfade mit /
- Sonderformen sind
  - / die Wurzel (XML Wurzel ist deren Nachfolger)
  - // alle Nachfolger
  - .. direkter Vorgänger
  - . aktuelles Element

Beispiel:

```
<xsl:template match="name">
   <strong><xsl:value-of select="."></strong>
</xsl:template>
```

Bedingungen

- Erweiterungsoperatoren für Vergleiche: `=`, `!=`, `<`, `>`, `<=`, `>=`
  Vorsicht: `<` und `>` müssen als Literale mit Entitäten (`<` und `>`) umschrieben werden!

- Elemente mit einem bestimmten Wert:

```
match="//profession[.='physicist']"
```

- Finde Elemente mit einem bestimmten Attribut

```
match="//person[@id='p4567']"
```

- Finde Elemente mit einem unbestimmten Attribut

```
match="//name[middle_initial]"
```

- Dazu gelten die logischen Operatoren: `and`, `or`

ALLGEMEINE AUSDRÜCKE IN XPATH

- mathematische Operatoren: `+`, `-`, `*`, `div`, `mod`

```
<xsl:template match="person">
   <xsl:value-of select="@id" div 10">
</xsl:template>
```

- Zeichenketten (Strings)
- Bool'sche Werte und Ausdrücke (`true` und `false`)
- diverse Zugriffs- und Filterfunktionen:
  - `position()`, `first()`, `count()`
  - `starts-with(last_name, 'T')`, `substring-after()`, `substring()`
  - `string-length()`, `round()`, `floor()`, `ceiling()`