6. Mediendokumente

Weiterführende Literatur:
M. Knobloch, M. Kopp: Web-Design mit XML, dpunkt-Verlag 2001

Generische Auszeichnungssprache: Idee

- Auszeichnungssprache *(markup language)*:
  - Text und spezielle textuelle Information zur Darstellung
  - Tag-/Attribut-Syntax weithin bekannt durch HTML
- Idee: „Familie“ von Auszeichnungssprachen gleicher Basissyntax für verschiedenste Anwendungsgebiete
  - Web-Seiten-Formatierung (HTML)
  - Strukturierte Daten, z.B. Adressen, Briefe, Texttypen
  - Austauschformate für Textverarbeitung und andere Software
  - Standardformate für Grafik, Multimedia-Präsentationen, ...
- Vorteile:
  - Lesbarkeit durch Mensch und Maschine
  - Trennung von Inhalt und Präsentation
  - Automatische syntaktische Überprüfungen
  - Erweiterbarkeit durch Definition neuer Tags/Attribute
- Nachteil: Lange Texte

Geschichte: SGML, HTML, XML, XHTML

- 1967: GenCode-Komitee
  - Norman Scharpf, Trennung Inhalt-Layout
- 1969: Goldfarb, Mosher, Lorrie (IBM):
  - Generalized Markup Language GML
- 1978: ISO-Standard 8879
  - Standard Generalized Markup Language SGML
  - Erlaubt Definition beliebiger *Dokumenttypen*
  - Sehr komplex, Verbreitung vorwiegend im akademischen Bereich und in der Definition weiterer Standards
- 1989: Berners-Lee, Cailleau
  - Hypertext Markup Language HTML
  - HTML ist ein spezieller Dokumenttyp von SGML
- 1998: WWW Consortium (W3C)
  - eXtensible Markup Language (XML)
  - Teilsprache von SGML
- 1999: Reformulierung von HTML als XML-Dokumenttyp
  - eXtensible Hypertext Markup Language XHTML
  - Etwas strengere Syntax als übliches HTML

XML-Dokumente

- Prolog:

```
<?xml version="1.0"?>
```

- Dokumenttyp:

```
<?xml >
```

- Tag- und Attribut-Syntax wie in HTML
  - Jedes geöffnete Tag muss explizit geschlossen werden.

```
<XY> ... <XY/>
```

  - Leere Tags müssen mit „/>“ enden

```
<BR/>
```

  - Jedes XML-Dokument hat genau ein Wurzel-Element *(root)*
  - Strenge hierarchische Schachtelung von Tags
  - Attributwerte immer in doppelten Anführungszeichen

```
<XY A="..."> ... <XY/>
```

  - Keine doppelten Attributwerte

Document Type Definition (DTD)

- Festlegung der zulässigen Werte für Tags, Attribute etc. in den zugehörigen XML-Dokumentdateien
- Meist in separater Datei, kann aber auch Bestandteil eines XML-Dokuments sein
- Zwei verschiedene Syntax-Alternativen:
  - klassische DTD-Syntax (hier beschrieben)
  - "XML Schema" (siehe später)
- Wichtigste Deklarationen in DTDs:
  - ELEMENT: Element (Dokument-Tag)
  - ATTLIST: Attributliste für ein Element
  - ENTITY: Abkürzung für komplexes Element
  - NOTATION: Datentyp-Deklaration
- Eingebaute Datentypen in XML:
  - PCDATA (Parsed Character Data) - vom Parser analysierte Zeichenreihe
  - weitere Datentypen zum Einsatz z.B. in Attributen

ELEMENT-Deklaration in DTD

- Syntax:

  <!ELEMENT *Elementname* (*Inhaltsbeschreibung*)>

- *Elementname:*
  - Name muss mit Buchstaben oder Unterstrich beginnen
  - Gross- und Kleinschreibung wird unterschieden
- *Inhaltsbeschreibung:*
  - Struktur des Inhalts zwischen Start- und End-Tag
  - Referenziert weitere Elemente oder eingebaute Datentypen
  - Reguläre Ausdrücke:

Komma-Liste:	Sequentielle Abfolge
A | *B* :	Alternativen
A ?	Optional
A +	Mindestens einmal
A *	Belliebig oft

Beispiel: Dokumenttyp „Folien“ V.1

- Eine (sehr einfache) Folie einer Präsentation hat folgende Bestandteile
  - Titel: Zeichenreihe
  - Liste von Themen
    - » Thema: Zeichenreihe

- Als DTD formal notiert:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT folie (titel, themenliste)>
<!ELEMENT titel (#PCDATA)>
<!ELEMENT themenliste (thema*)>
<!ELEMENT thema (#PCDATA)>
```

Beispiel in Anlehnung an Knobloch/Popp

Beispiel: Dokument des Typs „Folien“ V.1

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE folie SYSTEM "folien1.dtd">
<folie>
    <titel>XML-Übersicht</titel>
    <themenliste>
        <thema>Was sind DTDs?</thema>
        <thema>Struktur einer XML-Datei</thema>
    </themenliste>
</folie>
```

Wohlgeformtheit und Gültigkeit

- Ein XML-Dokument ist *wohlgeformt (well-formed)*, wenn es den allgemeinen Regeln der XML-Syntax genügt.
  - Beispiel (*well-formed*, aber nicht *valid*):

```
<folie>
    <ttel/>
    <tlist>
        <thm>Was sind DTDs?</thm>
        <tha>Struktur einer XML-Datei</tha>
    </tlist>
</folie>
```

- Ein XML-Dokument ist *gültig (valid)*, wenn es der angegebenen Document Type Definition entspricht.
- *Erweiterbarkeit:* Anwendungen können mit nur wohlgeformten oder nur teilweise gültigen Dokumenten arbeiten (z.B. zusätzliche herstellerspezifische Tags – werden im Zweifelsfall ignoriert).

Spezielle Inhaltsmodelle für ELEMENT

- ANY:
  - Erlaubt beliebige Zeichenreihen
  - Einschliesslich Markup, d.h. weiteren Tags im Inhalt
  - Beispiel:

```
<!ELEMENT titel ANY>
<titel><thema>XML-Übersicht</thema></titel>
```

- EMPTY:
  - Verlangt leeren Inhalt
- NDATA:
  - Inhalt sind binär codierte Daten
- Gemischte Daten (Literal und Element):
  - PCDATA-Angabe muss immer vorne stehen, z.B. in

```
<!ELEMENT titel (#PCDATA | thema)*>
```

ATTLIST-Deklaration in DTD

- Syntax:
  <!ATTLIST *Elementname Attributdefinition*+ >
- *Attributdefinition*:
  ***Attributname Attributtyp Standardwert* [*Festwert*]**
- *Attributtyp:*
  - Angabe eines Datentyps
    - » CDATA: Character Data, d.h. Zeichenreihe (nicht analysiert)
    - » ID: Eindeutiger Bezeichner für Verweise im Dokument
    - » IDREF: Verweis auf einen Bezeichner (vom Typ ID)
  - Explizite Werteliste (ohne Anführungszeichen!)
    - » (*Wert1* | *Wert2* | ....)
- *Standardwert:*
  - Angabe eines konkreten Werts: Default-Wert, Attribut ist optional
  - #IMPLIED: Attribut ist optional ohne Defaultwert
  - #REQUIRED: Attribut muss angegeben werden
  - #FIXED: Attribut muss immer mit dem selben Wert angegeben werden (der als *Festwert* angegeben ist)

Beispiel: Dokumenttyp „Folien“ V.2

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT folien (folie*)>
<!ATTLIST folien sprache (de | en)  "de">
<!ELEMENT folie (titel, themenliste)>
<!ATTLIST folie erstellt CDATA #REQUIRED>
<!ATTLIST folie autor CDATA #IMPLIED>
<!ATTLIST folie ident ID #REQUIRED>
<!ATTLIST folie sieheAuch IDREF #IMPLIED>
<!ELEMENT titel ANY>
<!ELEMENT themenliste (thema*)>
<!ELEMENT thema (#PCDATA)>
```

Beispiel: Dokument des Typs „Folien“ V.2

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE folien SYSTEM "folien2.dtd">
<folien sprache="en">
    <folie erstellt="04.06.2003" ident="f1">
        <titel>Attribute in XML</titel>
        <themenliste>
            <thema>Deklaration in DTD</thema>
            <thema>Verwendung in XML-Dokument</thema>
        </themenliste>
    </folie>
    <folie erstellt="03.06.2003" ident="f2">
        <titel>Identifikatoren</titel>
        <themenliste>
            <thema>Eindeutigkeit</thema>
        </themenliste>
    </folie>
</folien>
```


Elemente vs. Attribute

- Zusatzinformation zu Elementen als Unterelemente oder Attribute?

```
<buch>
   <isbn>3-932588-96-7</isbn>
   <autor>Fritz Müller</autor>
   ...
</buch>
```

oder

```
<buch
  isbn="3-932588-96-7"
  autor="Fritz Müller">
   ...
</buch>
```

- Unterelemente
  - sind leichter zu lesen
  - ermöglichen Wiederholungen
  - können weiter in Elemente untergliedert werden
  - können auch in anderem Kontext genutzt werden
- Attribute
  - sind leicht zu überprüfen
  - erlauben keine Wiederholungen
  - verringern die Hierarchietiefe
  - binden die Zusatzinformation eng an das zugehörige Element

Abkürzungen mit (internen) Entities

- (Interne) Entity
  - Abkürzung für komplexe XML-Konstruktion
  - Paar Name – Inhalt
- Allgemeine Entities
  - kürzen den Inhalt des Dokuments ab
  - Syntax:

```
<!ENTITY Entityname Entityinhalt >
```

- Parameter-Entities
  - Abkürzungsmechanismus innerhalb der DTD
  - Syntax:

```
<!ENTITY % Entityname Entityinhalt >
```


Beispiele für Entities

- Allgemeine Entities:
  - DTD:

```
<!ENTITY einleitung "<thema>Einleitung</thema>">
<!ENTITY schluss "<thema>Zusammenfassung und
   Ausblick</thema>">
```

  - XML-Dokument:

```
<themenliste>
  &einleitung;
   <thema>Kapitel 1</thema> ...
</themenliste>
```

- Parameter-Entities:
  - DTD:

```
<!ENTITY % xa "xml.apache.org">
<!ENTITY xalink "<link href='http://%xa; name='%xa;'/>">
```

Externe Entities

- Angabe einer externen Datei, auf die zugegriffen werden muss
  - Schlüsselwort SYSTEM
- Ungeparste Externe Entities:
  - besonders relevant z.B. zur Einbindung von Multimedia-Daten
  - NDATA: Signalisiert dem Parser, dass externes Hilfsprogramm zu verwenden ist
  - Separate NOTATION-Anweisung zur Definition des Datentyps
- Beispiel:

```
<!ENTITY meinFOTO SYSTEM "xyz.gif" NDATA GIF89A>
```

  - wobei Datentyp `GIF89a` separat definiert:

```
<!NOTATION GIF89a SYSTEM "xyz.gif" NDATA "gif_viewer.exe">
```

  - Alternativ nur Angabe zur Spezifikation (Schlüsselwort PUBLIC) in der NOTATION-Anweisung

Beispiel: Geräusch beim Blättern in Folien

- In der DTD:

```
<!NOTATION WAV SYSTEM "player.exe">
<!ENTITY weiter SYSTEM "raschel.wav" NDATA WAV>
```

- Im XML-Dokument:

```
<folie erstellt="04.06.2003" ident="f1"
   blaetterTon="weiter">
  <titel>Attribute in XML</titel>
  <themenliste>
    <thema>Deklaration in DTD</thema>
    <thema>Verwendung in XML-Dokument</thema>
  </themenliste>
</folie>
```

Namespaces

- Mehrdeutige Tags:
  - Tags können mit XML von jeder Person auf der Welt unabhängig definiert werden!
  - Eindeutigkeit nur lokal sicherzustellen
  - Was passiert beim Mischen von XML-Information aus verschiedenen Quellen?
- Namespace-Deklaration
  - Syntax:

    <*Tagname* xmlns:*Namensraumname*="*URI*" ...>
  - Definiert *Namensraumname* als zusätzliches Unterscheidungsmerkmal für alle Tags, die im Dokumentenbaum unterhalb von *Tagname* liegen
  - *URI* definiert den Urheber des Namensraums.
    Es wird NICHT über das Netz zugegriffen; wichtig ist nur die weltweit eindeutige Identifikation.
  - *Namensraumname* wird als *Präfix* für untergeordnete Tags verwendet:
    Namensraumname* : *Tag
  - Spezialfall: Default-Namensraum (für Tags ohne Präfix) deklariert durch
    <*Tagname* xmlns="*URI*" ...>

Namespaces: Beispiel

- Im XML-Dokument:

```
<folien sprache="en" xmlns="http://www.mimuc.de"
    xmlns:categXML="http://www.topicsXML.org">
...
<thema>
  Standard nach <categXML:thema>W3C</categXML:thema>
</thema
```

- Leider keine automatische Integration mit der DTD-Deklaration
  - Namespace-Attribute und -Elemente müssen in DTD explizit definiert werden

15 Gründe, nicht mit DTDs zufrieden zu sein

- DTDs haben eigene Syntax
- DTDs und XML 1.0 keine getrennten Standards
- Typsystem beruht nur auf Zeichenreihen
- Keine anderen komplexen Typen als Aufzählungen
- Mischung von Zeichenreihen- und strukturierten Werten störanfällig
- Namensräume nicht integriert
- Keine Modularität; keine Wiederverwendung
- Keine Unterstützung für Dokumenttyp-Evolution
- Verwendung von Leerraum kaum beschreibbar
- Keine Konzepte zur Selbst-Dokumentation
- Keine kontextabhängigen Deklarationen (in der Praxis zu "allgemeine" DTDs)
- ID-Attribute nicht sehr leistungsfähig
- Keine Default-Werte für Elemente
- Keine Möglichkeit, Aussagen für alle Attribute oder Elemente zu machen
- Default-Werte können nicht separat von der Syntax definiert werden.

ANDERS MØLLER & MICHAEL I. SCHWARTZBACH
http://www.brics.dk/~amoeller/XML/schemas/dtd-problems

XML Schema: Idee

- Ersatz von XML DTDs durch ein spezielles XML-Dokumentformat
  - Ermöglicht Formen der reflexiven Definition: Schemasprache in sich selbst definierbar
  - Erlaubt homogene Werkzeuge für Schemata und Dokumente
- Datentypkonzept
  - Vielzahl eingebauter primitiver Datentypen (z.B. Zahlen)
  - Strukturierte Datentypen *(complex datatype)*
- Strukturierter Aufbau
  - Vererbung auf Schema-Ebene
- Verbesserte Dokumentationsunterstützung

- Leider: Sehr komplex
  - Mehrere hundert Seiten Spezifikation!
  - Deshalb Fortschritt im praktischen Einsatz nur sehr langsam

Beispiel: XML Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
<xs:element name="folie" type="folie_type"/>
<xs:element name="titel" type="xs:string"/>
<xs:element name="thema" type="xs:string"/>
<xs:complexType name="folie_type">
    <xs:sequence>
        <xs:element ref="titel"/>
        <xs:element ref="thema" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
</xs:schema>
```


Beispiel: XML-Dokument basierend auf Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<folie xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
 xsi:noNamespaceSchemaLocation="folie1.xsd">
 <titel>XML-Übersicht</titel>
 <thema>Was sind DTDs?</thema>
 <thema>Struktur einer XML-Datei</thema>
</folie>
```

XML Parser APIs: DOM und SAX

- XML Parser API:
  - XML-Dateien werden typischerweise in individuell geschriebenen Programmen weiterverarbeitet
  - Besonders geeignet als Austauschformat
  - Parser API ermöglicht leichten Zugriff auf XML-Daten z.B. aus einem Java-Programm
  - Standard-APIs für XML Parser in CORBA spezifiziert
- DOM (Domain Object Model)
  - Repräsentation eines XML-Dokuments in Baumform ("Abstrakter Syntaxbaum")
  - Details siehe später
- SAX (Simple API for XML)
  - Verwendet einen Ereignismechanismus, um beim Parsen von XML-Dateien ein Benutzerprogramm aufzurufen
  - Besser geeignet für sehr grosse Dateien oder den Zugriff auf nur einen kleinen Teil der enthaltenen Information

6. Mediendokumente

Weiterführende Literatur:
M. Knobloch, M. Kopp: Web-Design mit XML, dpunkt-Verlag 2001

CSS und XML

- XML-Dateien enthalten „reinen Inhalt“
  - Zur Anzeige z.B. im Browser Zusatzangaben nötig
- Alternative Wege von einer XML-Datei zu einer Browseranzeige:
  - Cascading Style Sheets
  - Transformation in HTML-Text (z.B. mit XSLT, siehe später)
  - Kombination beider Ansätze

- Cascading Style Sheets
  - Separate Datei(en) mit Formatierungsangaben
  - Anbindung an XML-Dokument durch eine sogenannte *processing instruction* (PI):

```
<?xml-stylesheet type="text/css" href="folienstyle.css"?>
```


Beispiel: CSS-Datei für Dokumenttyp „Folien“ V.2

```
folie {
    font-family:sans-serif
}

titel {
    display:block; padding-top:10pt;
    font-size:200%; font-weight:bold; color:blue
}

thema {
    display:block; padding-left:30pt; padding-top:10pt;
    font-size:150%
}
```

Attribute in XML

- Deklaration in DTD
- Verwendung in XML-Dokument

Identifikatoren

Fortgeschrittene Konzepte in CSS

- Pseudo-Formate:
  - z.B. **`display:none`** zum Ausblenden (nicht darstellen)
- Pseudo-Elemente:
  - z.B. **`:first-letter`**, **`:first-line`** zur speziellen Formatierung von Zextteilen
  - z.B. **`:before`**, **`:after`** zum Modifizieren von Texten bei der Anzeige
- Pseudo-Klassen
  - z.B. **`:hover`**, **`:focus`**, **`:active`** zur Darstellung abhängig von Benutzeraktionen
- Kontextabhängige Formatierung
  - z.B. für Elemente abhängig von bestimmten Attributwerten
  - z.B. für Unterelemente abhängig von den im Dokument vorhandenen Oberelementen
- Strukturierung von Formatierungsinformation
  - Vererbung und verschiedene Formen zur Einbindung von Stylesheets