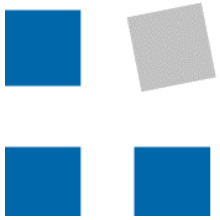


HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

XML

Teil 2: DTD

Abteilung Informatik – WS 02/03

DTD = Document Type Definition

- Eine zu einem gegebenen XML-Dokument zugeordnete DTD beschreibt,
  - welche Elemente und welche Entitäten in diesem Dokument vorkommen können
  - was die Inhalte und die Attribute der Elementen des Dokuments sind

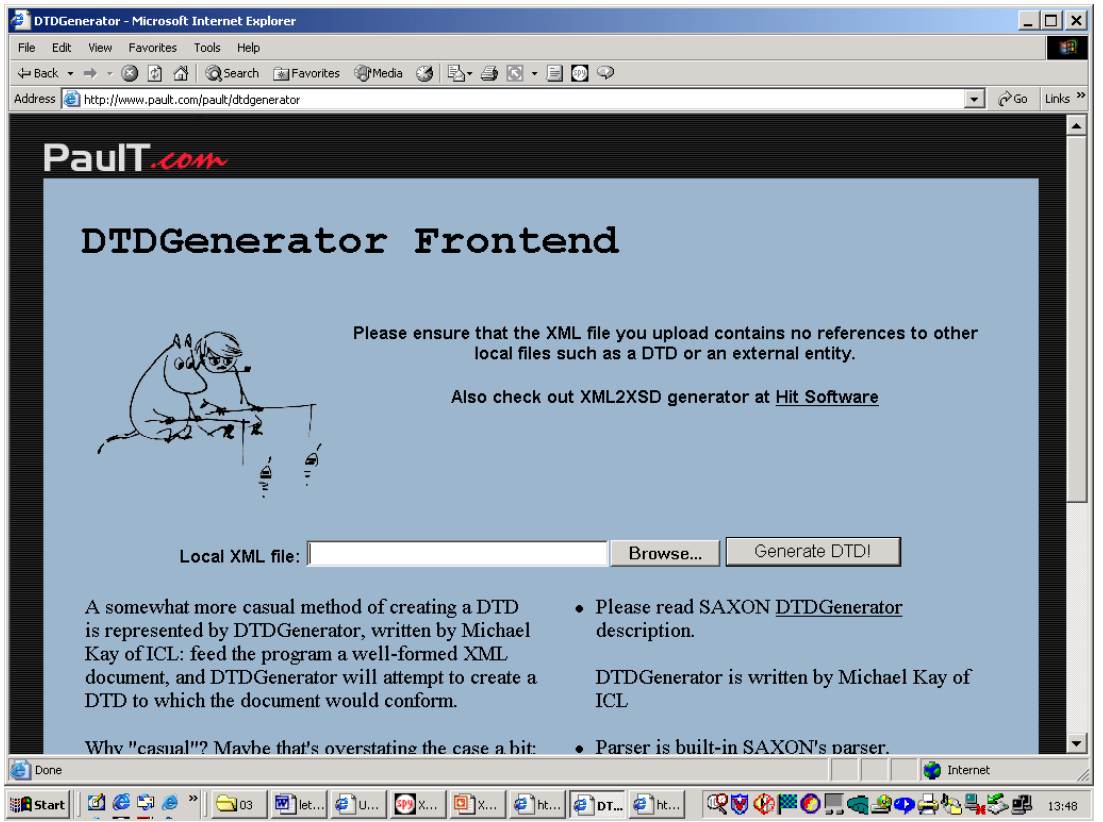
Parser

- **Gültigkeitsüberprüfung** (optional): Der Parser überprüft, ob die in der DTD definierten Anforderungen vom Dokument erfüllt sind
- **Gültigkeitsüberprüfungsprinzip**: Was nicht explizit erlaubt ist, ist verboten
- Wenn das Dokument eine Anforderung der entsprechenden DTD nicht erfüllt, gibt der Parser eine Fehlermeldung zurück. Die Anwendung (z.B. Browser) entscheidet, ob die Verarbeitung des Dokuments fortgesetzt werden soll
- Ein Dokument heisst **gültig**, wenn der Parser bei der Gültigkeitsüberprüfung keinen Fehler gemeldet hat. Sonst heisst das Dokument **ungültig**

DTD: Ein einfaches Beispiel

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

Automatische DTD-Generierung



http://www.pault.com/pault/dtdgenerator

Dokumenttyp-Deklaration

- Die **Dokumenttyp-Deklaration** (engl. Document Type Declaration) in einem XML-Dokument gibt eine Referenz zur entsprechenden DTD an
- Beispiel einer Dokumenttyp-Deklaration:

```
<!DOCTYPE person SYSTEM="http://ita.hsr.ch/xml/person.dtd">
```

- Der Wert des Attributs SYSTEM ist eine oder mehrere URI (Uniform Resource Indentifier)
  - In der Regel ist der Wert des Attribut SYSTEM eine (absolute oder relative) URL

Interne und externe DTD

- Die DTD kann ganz oder teilweise in der DTD-Deklaration integriert werden
- **Interne DTD-Untermenge**: Teil der DTD, welche in der Dokumettyp-Deklaration definiert ist
- **Externe DTD-Untermenge**: Teil der DTD, welche ausserhalb des XML-Dokuments definiert ist
- Das Attribut `standalone` der XML-Deklaration eines XML-Dokuments mit einer externen DTD-Untermenge muss den Wert `no` haben

XML-Dokument mit DTD: Beispiel (1/2)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE person [
  <!ELEMENT first_name (#PCDATA)>
  <!ELEMENT last_name (#PCDATA)>
  <!ELEMENT profession (#PCDATA)>
  <!ELEMENT name (first_name, last_name)>
  <!ELEMENT person (name, profession*)>
]>
<person>
      <name>
            <first_name>Alan</first_name>
            <last_name>Turing</last_name>
      </name>
      <profession>computer scientist</profession>
      <profession>mathematician</profession>
      <profession>cryptographer</profession>
</person>
```

XML-Dokument mit DTD: Beispiel (2/2)

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<!DOCTYPE person SYSTEM "C:\xml\ex1.dtd">
<person>
      <name>
            <first_name>Alan</first_name>
            <last_name>Turing</last_name>
      </name>
      <profession>computer scientist</profession>
      <profession>mathematician</profession>
      <profession>cryptographer</profession>
</person>
```

Parsing von externen DTD

- Ein Parser, der die Gültigkeit eines Dokuments überprüft, muss die externe DTD lesen
- Ein Parser, der nur die Wohlgeformtheit des Dokuments überprüft, muss nicht unbedingt die externe DTD lesen
- Konsequenz: Wenn die Deklarationen der externen DTD den Inhalt des Dokuments beeinflussen (z.B. Default Parameter setzen), hängt der Inhalt des XML-Dokuments vom Parser ab

Elementdeklarationen

- Syntax einer Elementdeklaration:

  `<!ELEMENT` *`name inhalt`*`>`

- *`name`*: Name des Elements
- *`inhalt`*: Inhaltsmodell
  - Name und Reihenfolge der Childelemente
  - Spezifikation der einzelnen Child-Elemente
- Die Reihenfolge der Elementdeklarationen spielt keine Rolle

#PCDATA

- Beispiel

```
<!ELEMENT first_name (#PCDATA)>
```

- Bedeutung: Das Element enthält nur geparste Zeichendaten (keine Child-Elemente)

Folgen

- Beispiel:

```
<!ELEMENT name (first_name, last_name)>
```

- Bedeutung: Das Element enthält eine geordnete Folge von Child-Elementen, deren Namen angegeben wird
- Die Folge kann aus einem einzigen Element bestehen
- Die Folge ist durch Klammern begrenzt
- Elemente werden durch Kommata getrennt

Folgen: Beispiel (1/2)

- Elementdeklaration

```
<!ELEMENT student (name, class)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT class (#PCDATA)>
```

- Gültig:

```
<student>
      <name> dupont </name>
      <class> 1998 </class>
</student>
```

Folgen: Beispiel (2/2)

- Ungültig:

```
<student>
   <class> 1998 </class>
   <name> dupont </name>
</student>
```

- Ungültig:

```
<student>
   <name> dupont </name>
</student>
```

- Ungültig:

```
<student>
   <name> dupont </name>
   <vorname> jean </vorname>
   <class> 1908 </class>
</student>
```

Anzahl der Elemente einer Folge

- Folgende (optionale) Suffixen, welche nach dem Elementname des betroffenen Child-Elements angegeben werden müssen, bestimmen die Anzahl der Elemente in einer Folge:
  - `?` : 0 oder 1 Element ist erlaubt
  - * : n Elemente sind erlaubt, $n \geq 0$
  - `+` : n Elemente sind erlaubt, $n \geq 1$
- Beispiel:

```
<!ELEMENT mannschaft (name, spieler+) >
```

Auswahl

- Beispiel:

```
<!ELEMENT Zahlungsbetrag (Kreditkarte|Bar)>
```

- Bedeutung: Das Element `Zahlungsbetrag` kann entweder das Element `Kreditkarte` oder das Element `Bar` enthalten

Gruppieren mit Klammern

- Beispiel:

```
<!ELEMENT name ((first_name, middle_name,
last_name) | (first_name, last_name)  |
last_name)>
```

- Jedes Klammerpaar kombiniert mehrere Elemente, so dass die Kombination während der Überprüfung als einzelne Einheit behandelt wird
- Suffixen könne auch mit Klammer verwendet werden

```
<!ELEMENT liste (name,(buch|artikel)+)>
```

Gemischter Inhalt

- Beispiel
  - XML-Dokument

```
<text><fett>Definition.</fett> XML ist eine <kursiv>Meta-
Auszeichnungssprache</kursiv>, die erlaubt, die
benötigten Tags nach Bedarf zu generieren.</text>
```

  - Entsprechende DTD

```
<!ELEMENT fett (#PCDATA)>
<!ELEMENT kursiv (#PCDATA)>
<!ELEMENT text (#PCDATA | fett | kursiv)*>
```

- Bemerkungen
  - Das ist die einzige Methode, um einen gemischten Inhalt anzugeben
  - Gemischte Inhalte sind in narrativen Texten üblich (Webseiten, Zeitungsartikel, usw.)
  - Datenzentrische Dokumente sollten gemischte Inhalte vermeiden

Leere Elemente

- Leere Elemente (d.h. Elemente ohne Inhalt) werden mit dem Schlüsselwort EMPTY deklariert. Zum Beispiel:

```
<!ELEMENT BR EMPTY>
```

ANY-Inhaltsmodell

- Beispiel:

```
<!ELEMENT Weltall ANY>
```

- Bedeutung: Beliebige andere Elemente sowie geparste Zeichendaten können Child-Elemente sein.

Attributdeklaration

- Attribute werden in einer `ATTLIST`-Deklaration deklariert.
- Mehrere Attribute können in einer ATTLIST-Deklaration deklariert werden
- Beispiel:

```
<!ELEMENT book EMPTY>
<!ATTLIST book
  author CDATA "Louis Dupont"
  editor CDATA "Editout">
```

Attributswerte und -Typen

- Es gibt 10 Attributstypen in XML

  CDATA

  NMTOKEN, NMTOKENS

  Enumeration

  ENTITY, ENTITIES

  ID, IDREF, IDREFS

  NOTATION

Vorgabewerte

- Der Default-Wert ist vorgegeben

```
<!ATTLIST AUTHOR NAME CDATA "Dupont">
```

- Der Wert des Attributs ist konstant und kann nicht verändert werden: Schlüsselwort `#FIXED` mit Attributswert

```
<!ATTLIST AUTHOR NAME CDATA #FIXED "Dupont">
```

- Das Attribut ist optional: Schlüsselwort `#IMPLIED`

```
<!ATTLIST AUTHOR NAME CDATA #IMPLIED>
```

- Das Attribut ist verlangt: Schlüsselwort `#REQUIRED`

```
<!ATTLIST AUTHOR NAME CDATA #REQUIRED>
```

Attributstyp CDATA

- Attributswert = beliebige Folge von Zeichen
- 1. Ausnahme: Die Zeichen < und " dürfen nicht verwendet werden (sondern `<` und `&quot;`)
- 2. Ausnahme: Das Zeichen & muss mit `&` referenziert werden, falls es nicht am Anfang einer Entitätsreferenz steht
- `CDATA`: Geeigneter Attributstyp für URI, freie Texte, etc.
- Beispiel. Dem Element

```
<BILD GROESSE="10&quot; "/>
```

entspricht die Deklaration

```
< !ATTLIST BILD GROESSE CDATA #REQUIRED>
```

Attributstyp NMTOKEN

- Tokens-Namen sind ähnlich wie XML-Namen
- Wie XML-Namen:
  - Alphanumerische Charaktere sind zulässig
  - `_` `-` `.` und `:` sind zulässig
  - Leerzeichen sind nicht zulässig
- Nicht wie XML-Namen:
  - Das erste Zeichen darf irgendein zulässiger Charakter sein
- Beispiel: `.cshrc` ist ein zulässiger Tokens-Name, aber kein zulässiger XML-Name

Attributstyp NMTOKENS

- NMTOKENS: ein oder mehrere Namens-Token, die durch ein Leerzeichen getrennt sind
- Beispiel. Dem Element

```
<LISTE MEMBERS="Jules Jean Jacques">
Liste A </LISTE>
```

entspricht die Deklaration

```
<!ATTLIST LISTE MEMBERS NMTOKENS
                #REQUIRED>
```

Attributstyp Enumeration

- Liste möglicher Werte für das Attribut
- Jeder mögliche Wert muss ein Namens-Token sein
- Syntax:

$$(wert_1|wert_2|...|wert_n)$$

- Beispiel: Dem Element:

```
<TOPIC VISIBLE = "true">Thema</TOPIC>
```

entspricht die Deklaration:

```
<!ATTLIST TOPIC VISIBLE (true|false) #REQUIRED>
```

Attributstyp ID

- Der Wert eines Attributs vom Typ ID ist ein XML-Name, welche im ganzen Dokument einzigartig ist (d.h. kein weiterer Attribut vom Typ ID darf denselben Wert haben)
- Beispiel. Mit der DTD:

```
<!ELEMENT LISTE (RECHNUNG|BESTELLUNG)*>
<!ELEMENT BESTELLUNG (#PCDATA)>
<!ELEMENT RECHNUNG (#PCDATA)>
<!ATTLIST BESTELLUNG NUMMER ID #IMPLIED>
<!ATTLIST RECHNUNG REFERENZ ID #IMPLIED>
```

ist das folgende Dokument ungültig:

```
<LISTE>
  <BESTELLUNG NUMMER="A1299">Laptop</BESTELLUNG>
  <RECHNUNG REFERENZ="A1299">Buch</RECHNUNG>
</LISTE>
```

Attributstyp IDREF (1/2)

- Der Wert eines Attributs vom Typ IDREF ist die ID eines anderen Elements im Dokument
- Der Typ IDREF wird benötigt, um Verbindungen zwischen Elementen herzustellen, die nicht durch die Baumstruktur des Dokuments vorgegeben werden
- Beispiel:

```
<!ATTLIST employee social_security_number ID #REQUIRED>
<!ATTLIST project project_id ID #REQUIRED>
<!ATTLIST team_member person IDREF #REQUIRED>
<!ATTLIST assignment project_id IDREF #REQUIRED>
```

Attributstyp IDREF (2/2)

Gültiges Dokument für die obige DTD:

```
<project project_id="p1">
  <goal>Develop Strategic Plan</goal>
  <team_member person="ss078-05-1120"/>
  <team_member person="ss987-65-4320"/>
</project>
<project project_id="p2">
  <goal>Deploy Linux</goal>
  <team_member person="ss078-05-1120"/>
  <team_member person="ss9875-12-3456"/>
</project>
<employee social_security_number="ss078-05-1120">
  <name>Fred Smith</name>
  <assignment project_id="p1"/>
  <assignment project_id="p2"/>
 </employee>
 <employee social_security_number="ss987-65-4320">
   <name>Jill Jones</name>
   <assignment project_id="p1"/>
 </employee>
 <employee social_security_number="ss9875-12-3456">
   <name>Sydney Lee</name>
   <assignment project_id="p2"/>
 </employee>
```

Attributstyp IDREFS (1/2)

- Der Wert eines Attributs vom Typ IDREFS ist eine Liste von ID anderer Elemente im Dokument
- Der Typ IDREFS wird verwendet, wenn ein Attribut auf mehr als eine ID im Dokument verweisen muss
- Beispiel:

```
<!ATTLIST employee social_security_number ID #REQUIRED
                   assignments IDREFS #REQUIRED>
<!ATTLIST project  project_id ID #REQUIRED>
                   team IDREFS #REQUIRED>
```

Attributstyp IDREFS (2/2)

Gültiges Dokument für die obige DTD:

```
<project project_id="p1" team="ss078-05-1120 ss987-65-4320">
   <goal>Develop Strategic Plan</goal>
 </project>
 <project project_id="p2" team="ss078-05-1120 ss9875-12-3456">
    <goal>Deploy Linux</goal>
 </project>
 <employee social_security_number="ss078-05-1120" assignments="p1 p2">
   <name>Fred Smith</name>
 </employee>
 <employee social_security_number="ss987-65-4320" assignments="p1">
    <name>Jill Jones</name>
 </employee>
 <employee social_security_number="ss9875-12-3456" assignments="p2">
    <name>Sydney Lee</name>
 </employee>
```

Attributstyp ENTITY

- Mit dem Attributstyp `ENTITY` können externe Binärdaten (d.h. externe nicht geparste allgemeine Entitäten, z.B. ein Bild) in einem Dokument eingebunden werden
- Der Wert des `ENTITY`-Attributs entspricht der Namen einer allgemeinen Entität, die in der DTD deklariert ist, welche die Verknüpfung zu den externen Daten herstellt
- Beispiel. Die Deklarationen in der DTD:

```
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE SOURCE ENTITY #REQUIRED>
<!ENTITY LOGO SYSTEM "logo.gif">
```

entsprechen folgendem Element im XML-Dokument

```
<IMAGE SOURCE="LOGO">
```

Attributstyp ENTITIES

- Der Attributstyp ENTITIES verfügt über einen Wert, der aus mehreren ungeparsten Entitätsnamen besteht, die durch Leerzeichen voneinander getrennt sind
- Jeder Entitätsname referenziert eine externe Nicht-geparste-Datenquelle
- Beispiel. Die Deklarationen in der DTD:

```
<!ELEMENT SLIDESHOW EMPTY>
<!ATTLIST SLIDESHOW SOURCES ENTITIES #REQUIRED>
<!ENTITY PICTURE1 SYSTEM "picture1.gif">
<!ENTITY PICTURE2 SYSTEM "picture2.gif">
<!ENTITY PICTURE3 SYSTEM "picture3.gif">
```

entsprechen folgendem Element im XML-Dokument

```
< SLIDESHOW SOURCES=" PICTURE1 PICTURE2 PICTURE3">
```

Attributstyp NOTATION

- Der Wert eines Attributs vom Typ NOTATION ist der Name einer Notation, welche in der DTD deklariert ist
- Selten verwendeter Attributstyp
- Beispiel. Die folgenden Deklarationen...
  - definieren zwei Notationen für verschiedene Bildertypen
  - spezifizieren, dass jedes IMAGE-Element ein Attribut TYPE hat, das genau einen dieser Bildertypen hat

```
<!NOTATION gif SYSTEM "image/gif">
<!NOTATION tiff SYSTEM "image/tiff">
<!ATTLIST IMAGE TYPE NOTATION (gif|tiff) #REQUIRED>
```

Entitätsreferenzen (Wiederholung)

- Entität = Speichereinheit, die einen bestimmten Teil (wohlgeformter XML-Code, Text in anderer Form, Binärdaten) eines XML-Dokuments enthält
- Es gibt fünf vordefinierte Entitätsreferenzen. Sie beziehen sich auf Zeichen (Textentität)

`&` für `&`

`<` für `<`

`>` für `>`

`&quot;` für `"`

`&apos;` für `'`

Entitäten

- **Interne** Entitäten sind vollständig innerhalb des Dokuments definierte Entitäten
- **Externe** Entitäten beziehen ihren Inhalt aus anderen Quellen, die mit einer URL angegeben werden
- **Geparste** Entitäten enthalten einen syntaktisch korrekten XML-Text
- **Ungeparste** Entitäten enthalten entweder Binärdaten oder einen Nicht-XML-Text (z.B. E-mail-Nachricht)

Interne allgemeine Entitäten

- Entitätsreferenzen können in der DTD definiert werden
- Syntax einer **internen allgemeinen** Entitätsreferenz

```
<!ENTITY name "Ersetzungstext">
```

- Beispiel:

```
<!ENTITY path "C:\EigeneDateien\Programs\XMLParser.exe">
```

- Der Ersetzungstext kann auch ein wohlgeformtes XML-Dokument sein

Externe geparste Entitäten

- Syntax: ENTITY-Deklaration mit SYSTEM-Schlüsselwort, z.B.

```
<!ENTITY footer SYSTEM "http://www.hsr.ch/footer.xml">
```

- Nach Einbindung der externen geparsten Entität muss das XML-Dokument wohlgeformt sein. Daraus folgt:
  - Jedes Element, das in der externen Entität beginnt, muss auch in der externen Entität enden
  - Die externe Entität hat keinen Prolog (XML-Deklaration, Dokumenttyp-Deklaration)
- Eine externe geparste Entität kann eine Text-Deklaration haben. Beispiel:

```
<?xml version="2.0" encoding="ISO-8559-1">
```

(`version`: optional; `encoding`: verlangt; kein `standalone`-Deklaration)

Externe ungeparste Entitäten

- ENTITY-Deklaration mit dem Namen und die URI der externen Entität. Beispiel:

```
<!ENTITY logo SYSTEM "http://www.hsr.ch/logo.gif" NDATA gif>
```

- Die NDATA-Deklaration spezifiziert das Datenformat
- Beispiel von Einbindung eines ungeparsten Elements in ein XML-Dokument. Der DTD

```
<!ELEMENT image EMPTY>
<!ATTLIST image source ENTITY #REQUIRED>
<!ENTITY logo SYSTEM "http://www.hsr.ch/logo.gif "
NDATA gif>
```

  entspricht folgenden XML-Code

```
<image source="logo"/>
```

- Alternative, um Nicht-XML-Dokumente einzubinden: URL mit XLink

Notation

- Problemstellung:Eine Anwendung muss das Format der Daten wissen, damit sie sie lesen und anzeigen kann
- Eine Notation beschreibt das Format eines ungeparsten Dokuments
- Syntax der `NOTATION`-Deklaration in der DTD:

```
<!NOTATION name SYSTEM "externalID">
```

- Beispiel: Verwendung von MIME-Typen als „Identifier“ in einer `NOTATION`-Deklaration:

```
<!NOTATION gif SYSTEM "image/gif">
```

- Es gibt keine Regelung für die Auswahl des Identifiers

Parameter-Entität

- Allgemeine Entitätsreferenzen dürfen keinen Text einfügen, der nur als Teil der DTD und nicht als Teil des Dokumentinhalts verwendet wird. Folgender Code ist zum Beispiel ungültig:

```
<!ELEMENT TRANSACTION EMPTY>
<!ENTITY NR
  "789271273497237597356572658726348756238975623 65">
<!ATTLIST TRANSACTION REFERENZ CDATA "&NR;">
```

- Parameter-Entitäten dürfen nur in der DTD, nicht im Dokumentinhalt erscheinen

Parameter-Entitätsreferenz

- Deklaration:

```
<!ENTITY % name "Ersetzungstext">
```

- Referenz im XML-Dokument:

```
%name;
```

Parameter-Entitätsreferenzen (1/2)

Die folgenden Codes sind äquivalent:

example1.dtd

```
<!ELEMENT EINTRAG (NAME,VORNAME,ADRESSE,TELNUMMER)>
<!ELEMENT PERSON (NAME,VORNAME,ADRESSE,TELNUMMER)>
<!ELEMENT KUNDE (NAME,VORNAME,ADRESSE,TELNUMMER)>
<!ELEMENT KONTAKT (NAME,VORNAME,ADRESSE,TELNUMMER)>
```

example2.dtd

```
<!ENTITY % LIST "(NAME,VORNAME,ADRESSE,TELNUMMER)">
<!ELEMENT EINTRAG %LIST;>
<!ELEMENT PERSON %LIST;>
<!ELEMENT KUNDE %LIST;>
<!ELEMENT KONTAKT %LIST;>
```

Parameter-Entitätsreferenzen (2/2)

- Parameter-Entitätsreferenzen können neu definiert werden. Zum Beispiel ist folgendes XML-Dokument gültig

```
<!DOCTYPE EINTRAG SYSTEM "example2.dtd" [
<!ENTITY % LIST "(NAME, VORNAME)">]>
<EINTRAG>
   <NAME>Dupont</NAME>
   <VORNAME>Jean></VORNAME>
</EINTRAG>
```

Referenzen

- E. Harold, W. Means: *XML in a Nutshell*, Kapitel 3