DBAI

181135 VU Semistrukturierte Daten 1

Namensräume und Dokumenttyp-Definitionen
11.10.2005

Reinhard Pichler

Inhalt

- Namespaces – Namensräume für XML
- Schemata – Schemasprachen für SSD
- DTD – Document Type Definition
- Validierung (mit Xerces und IE6)

- Namen in globalem Kontext
- Namespaces verhindern Namenskonflikte
  - zwei Leute erweitern ein Dokument in inkompatibler Art
  - Mechanismus um mit dieser Erweiterbarkeit umzugehen
  - man möchte nicht die Flexibilität von XML verlieren
- Verwendung von Standards
  - XML: lang, space, ...
  - XML Schema: element, sequence, ...
  - XSLT: element, value-of, ...
  - XLink: title, role, ...

Beispiel: Erweiterung

Studenten führen Bewertungsschema für Lehrveranstaltungen ein:

```
<veranstaltung>
    <titel>Semistrukturierte Daten 1</titel>
    <bewertung>naja...</bewertung>
</veranstaltung>
```

Herausgeber eines annotierten Vorlesungsverzeichnis fügt selbes Tag ein (mit anderer Bedeutung) :

```
<veranstaltung>
    <titel>Semistrukturierte Daten 1</titel>
    <bewertung>Wahlpflichtfach</bewertung>
</veranstaltung>
```

Problem: Kombination? Umbenennung?
Lösung: Namespaces

Namespaces

- ursprünglich Namespaces in eigener Recommendation
- jetzt Teil von XML 2nd Edition
- XML für Datenaustausch, daher sind eindeutige Identifier notwendig und sinnvoll
  - URLs (allg. URIs) eignen sich dafür, da gibt es schon Eindeutigkeit
  - URL dient nur als Identifier, keine Referenz!
  - für NS sind www.dbai.tuwien.ac.at und www.DBAI.tuwien.ac.at nicht identisch
- Beispiel-Namespaces:
  - HTML 4.0: `http://www.w3.org/TR/REC-html40`
  - XML Schema: `http://www.w3.org/2001/XMLSchema`
  - XSLT: `http://www.w3.org/1999/XSL/Transform`

Beispiel: Namespaces

```
<?xml version="1.0"?>
<lehre xmlns:stud="http://www.stud.info/bew/1.4"
       xmlns:vorl="http://verkaeufer.com/vorl/1.0"
       xmlns="http://www.lehre.tuwien.ac.at/lehre/2.0">
...
 <veranstaltung vorl:jahr="2003">
    <titel>Semistrukturierte Daten 1</titel>
    <stud:bewertung>naja...</stud:bewertung>
    <vorl:bewertung>Wahlpflichtfach</vorl:bewertung>
 </veranstaltung>
...
</lehre>
```

Namespaces

- "Umbenennung"
  - Anstatt eines URI geben wir Abkürzung für den Identifier an
  - dient nur als Abkürzung (versch. Abkürzungen, die auf selben URI zeigen, definieren selben NS)
  - Namespace Präfix gefolgt von Elementname (oder auch Attributname)
  - getrennt durch Doppelpunkt
- zusätzlich Deklaration der Präfixe
  - im Element-Starttag als Attribut angegeben, gilt für Element selbst und alle Subelemente
  - assoziiert URI mit Präfix
  - sinnvoll da URIs immer eindeutig
  - xmlns Attribut ohne NS Angabe erklärt *Default Namespace*

Namespace Deklaration

- NS in Element deklariert
  - nur bindbar an Element selbst bzw. an geschachtelte Elemente
  - auch Attribute können NS aufweisen; Attribute ohne NS Präfix sind immer im leeren Namensraum
  - Nachfahren können NS überschreiben
  - Deklaration mittels Attribut `xmlns:`*`name`*
- Default Namespace
  - Elemente ohne NS Präfix (Attribute ohne NS Präfix sind im leeren NS)
  - Deklaration mittels Attribut `xmlns`
- Parser
  - NS aware: NS Abkürzung wird durch vollen URI ersetzt, Unterscheidung: qualifizierter Name, lokaler Name, NS Name
  - Nicht NS aware: Namespace Abkürzung als Teil des Namens, Namespace Definitionen als normale Attribute

Beispiel: Namespaces

```
<?xml version="1.0"?>
<lehre xmlns:stud="http://www.stud.info/bew/1.4"
       xmlns:vorl="http://verkaeufer.com/vorl/1.0"
       xmlns="http://www.lehre.tuwien.ac.at/lehre/2.0">
...
<veranstaltung xmlns="http://www.woanders.at/test">
   <titel>Semistrukturierte Daten 1</titel>
   <stud:bewertung>naja...</stud:bewertung>
   <vorl:bewertung>Wahlpflichtfach</vorl:bewertung>
 </veranstaltung>
...
</lehre>
```

liegen im neu definierten NS

Einschränkung: wäre veranstaltung ein Element von einem anderen NS wie z.B. stud:veranstaltung dürfte der Default NS nicht überschrieben werden

Namespaces

- URIs (Uniform Resource Identifier)
  - URLs (Uniform Resource Locator): Internetadressen nach „traditionellem“ Adressierungsschema, z.B.: http://...., ftp://...., mailto:.... von Ort des Dokuments abhängig oder nicht (PURL: permanent URL)
  - URNs (Uniform Resource Name): generische Namen, unabhängig von Ort des Dokuments, z.B. **`urn:ISBN:3-8274-0130-5`**
- Namen
  - Qualifizierter Name: **`stud:bewertung`**
  - Lokaler Name: **`bewertung`**
  - Namespace Name: **`http://www.stud.info/bew/1.4`**

Beispiel 1

Schemata

- Metainformationen über ein Dokument
  - Schema und Typisierungen
- Schema im Vorhinein definieren oder im Nachhinein den Daten auferlegen
  z.B. in XML Datenbanken geschieht beides
  - Kategorie, in die das Dokument fällt
  - Strukturindex der vorkommenden Pfade
- Dokumente als Instanzen einer Applikation
- Gültigkeit vs. Wohlgeformtheit
  - Gültigkeit: zusätzlich Übereinstimmung mit einem Schema verlangt

XML Schemasprachen

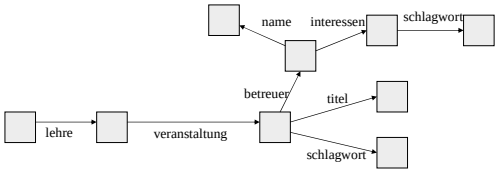
- DTD
  - Teil der XML-Recommendation
  - keine XML Syntax
  - eingeschränkte Möglichkeiten
- XML Schema
  - W3C Recommendation
  - XML Syntax
  - erweiterte Möglichkeiten
- Weitere Schemasprachen:
  - DCD (Document Content Description): Microsoft/IBM
  - SOX (Schema for Object Oriented XML)
  - RelaxNG: „Kompromiss“ zwischen DTDs und XML Schema.

Strukturgenerierung

- Typisierung von semistrukturierten Daten
- im Nachhinein wird ein passendes Schema aus den Daten abgeleitet, das Abfragen vereinfacht
- eine Art "Strukturindex"
- Formalismen für Strukturgewinnung
  - Logische Formeln: z.B.
    für alle "*veranstaltungen*" gibt es mindestens ein ....
  - Simulation: Aufbau eines Schema-Graphen
- Data Guides (Schemagraphen) werden in Lore verwendet

Data Guides

- Exakte Data Guides
  - Abstraktion der modellierten Daten
  - jeder unterschiedliche Pfad der DB kommt genau einmal im abstrahierten Graph G vor
  - zu jedem Pfad in G gibt es mindestens einen entsprechenden Pfad in DB
- Approximierte Data Guides
  - enthalten ev. zusätzliche Pfade



DTD

- XML vs. Datenbank
  - ohne DTD: XML selbstbeschreibend
  - nur semistrukturiert, keine starre Typendefinition
- Erstellen einer DTD
  - basierend auf Beispieldokumenten
  - basierend auf UML Diagrammen
- Deklarationen:
  - Elemente, Attribute, Entitäten, (Notationen)
- Formale Dokumentbeschreibung
  - mit kontextfreier Grammatik (nicht in XML-Format)
  - deterministisches Modell, z.B.: `(a,b)|(a,c)` verboten;
    statt dessen: `(a,(b|c))`

- wohlgeformtes vs. gültiges XML-Dokument
  - wohlgeformt: Parser überprüft nur XML Syntaxregeln
  - gültig: Parser validiert gegenüber DTD (oder einer anderen Sprache)
- Beispiele für DTD Standards
  (http://www.oasis-open.org/cover/xml.html#applications)
  - XHTML
  - MathML (Mathematical Markup Language)
  - CML (Chemical Markup Language)
  - RDF (Resource Description Framework)
  - ....

Dokument Typ Deklaration

- Einbettung in XML mittels DOCTYPE-Deklaration:
  `<!DOCTYPE ...>`
- direkt nach der XML Deklaration
- interner Teil vs externer Teil
  - intern: zwischen eckigen Klammern in der Deklaration
  - extern: in separater Entität, referenziert
- Mischung extern/intern möglich:
  - interne Definition hat Vorrang gegenüber einer externen
  - Elemente und Notationen: Mehrfachdefinitionen verboten
- „Standalone Dokument“ bei ausschließlich internem Teil:
  - optional: `<?xml version="1.0" standalone="yes"?>`

interne/externe DTD-Teilmenge

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- DBAI -->

<!DOCTYPE lehre SYSTEM "lehre.dtd"
   [
        <!ATTLIST veranstaltung jahr CDATA #REQUIRED>
   ]
>

<lehre>
  <veranstaltung jahr="2003">
    <titel>Semistrukturierte Daten 1</titel>
  ......
</lehre>
```

Externer Teil: System vs. Public

- System Identifier ist "lokaler" Identifier
  - d.h. nicht, dass er notwendigerweise eine lokale URL haben muss, z.B. `<!DOCTYPE seminar SYSTEM "http://www.seminar.se/se.dtd">`
  - für alle Nichtstandarddokumentarten
- Public Identifier
  - es muss dem XML Prozessor bekannt sein, was damit zu tun ist
  - für bekannte Dokumentarten
  - muss bestimmte Syntax befolgen (s.u.)
  - kann zweiten Wert aufweisen, der SYSTEM entspricht, wenn erster Wert nicht aufgelöst werden kann.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "xhtml11-flat.dtd">
```

Erster Charakter ist + wenn ISO, sonst –
Zweiter Teil ist der DTD Eigentümer
Dritter Teil ist DTD Beschreibung
Vierter Teil ist Sprache

Elementdeklaration

- Ein Element kann andere Elemente enthalten oder nur Text oder gemischten Inhalt
- Mögliche Inhaltsmodelle:
  - nur Text, nur Elemente, gemischt, beliebig, leer.
- Einschränkungsmöglichkeiten in DTDs
  - beliebiger Element-Inhalt
  - leerer Element-Inhalt
  - Vorkommen innerer Elemente
  - Zeichendaten (keine Typisierungen, nur Strings)

Elementdeklaration: Syntax

- Schlüsselwörter
  - #PCDATA ist Charakterisierung für Blattelemente (parsed character data)
  - EMPTY: leeres Element
  - ANY: beliebiger Inhalt (die vorkommenden Elemente müssen aber definiert sein in DTD)
- Auftretensindikatoren
  - reguläre Ausdrücke +,*,?
  - ohne: genau einmal
  - +: mindestens einmal und beliebig oft
  - *: 0-mal oder öfter
  - ?: 0-mal oder einmal
- Gruppierung
  - mit Klammern (...)

Elementdeklaration: Beispiele

```
<!ELEMENT lehre ANY>
Instanz:
<lehre>irgendetwas<sonst>und noch etwas</sonst></lehre>

<!ELEMENT lehre EMPTY>
Instanz:
<lehre/>

<!ELEMENT veranstaltung (name, jahr)>
Instanz:
<veranstaltung>
   <name>Seminar</name><jahr>2003</jahr>
</veranstaltung>
keine Instanz:
<veranstaltung>
   <jahr>2003</jahr><name>Seminar</name>
</veranstaltung>

<!ELEMENT lehre (#PCDATA)>
Instanz:
<lehre>Seminar</lehre>
keine Instanz:
<lehre>
   <jahr>2003</jahr><name>Seminar</name>
</lehre>
```

Elementdeklaration: Syntax

- Konnektoren
  - Sequenz: ","
  - Auswahl: "|"
- "," angeführte Elemente in genau dieser Reihenfolge!
- "|" nur eines darf vorkommen (exklusives oder)
- "|" iteriert mit * oder + erlaubt daher beliebige Reihenfolge (und öfteres Auftreten)
- verschachtelte Klammerung, z.B.
  `(lname, (fname | title))`

Elementdeklaration: Gemischter Inhalt

- Gemischter Inhalt vs. Element-Inhalt
  - Element content: nur Elemente enthalten
    Beispiel: `<!ELEMENT vorbesprechung (datum, zeit, ort)>`
- Mixed content
  - immer nur mit | trennen in DTD und * am Ende
    z.B. nicht erlaubt: `<!ELEMENT name (#PCDATA , fname , lname)*>`
- #PCDATA immer an erster Stelle spezifizieren
  - und nur mit | und * wenn ein mixed content drinnen steht!
    `<inhalt>Das ist <i>ein</i> Inhalt</inhalt>`
- #PCDATA darf nicht gemeinsam mit "," stehen
  z.B. nicht erlaubt: `<!ELEMENT lehre (#PCDATA, veranstaltung)>`

Elementdeklaration: Weitere Beispiele

```
<!ELEMENT lehre (veranstaltung+)>
```

Lehre ist eine Liste von Veranstaltungen (mindestens eine in diesem Beispiel).

```
<!ELEMENT buchtitel (deutsch | englisch* | italienisch)>
```

Eine Veranstaltung besteht entweder aus einer deutschen oder aus mehreren englischen oder aus einer italienischen Bezeichnung.

```
<!ELEMENT name (#PCDATA | fname | lname)*>
```

Ein Name besteht aus Vorname, Nachname, oder einer Mischung daraus mit gewöhnlichem Text.

```
<!ELEMENT veranstaltung ((name, jahr?)+)>
```

Liste von Namen/Jahren, wobei Jahrangabe optional ist.

Attributdeklaration

- Attribute: Name, Typ, Vorgabedeklaration
- Definition über Attributlisten
- entweder alle Attribute in einer Deklaration oder verstreut über mehrere Deklarationen
- Angabe von zugehörigem Element
- Die Reihenfolge ist egal

Attributdeklaration: Beispiele

```
<!ELEMENT zeit (#PCDATA)>
<!ATTLIST zeit sine-tempore (yes|no) "no">
```

Zeit enthält ein Attribut. Das Attribut sine-tempore kann "yes" oder "no" annehmen, wobei "no" der Defaultwert ist.

```
<!ATTLIST test href CDATA #REQUIRED>
```

Das href Attribut muß in test immer angegeben werden, und kann einen beliebigen Stringwert annehmen.

```
<!ATTLIST test xml:lang NMTOKEN #IMPLIED>
```

Wenn diese speziellen Attribute im Dokument vorkommen, dann müssen sie in bestimmter Weise (hier NMTOKEN und optional für xml:lang) in der DTD definiert werden.

Attributdeklaration: Syntax

```
<!ELEMENT zeit (#PCDATA)>
<!ATTLIST zeit sine-tempore (yes|no) "no">
```

- Elementname: "zeit"
- Attributname: "sine-tempore"
- Attributtyp (im Prinzip nur Stringtypen): Aufzählungstyp
- Vorgabedeklaration: Defaultwert "no"

Attributwerte können stärker beschränkt werden als Elementwerte.

Beispiel 2

Attributtypen

- CDATA (Character Data, String):
  - Zeichenkette in " " oder ' '
- Token-Typen:
  - Werte können auf verschiedene Arten eingeschränkt werden (s.u.)
- Aufzählungstyp:
  - Liste `(I|IL|IM|KB|KU|LA|LZ|RE|SZ)` ohne " "
  - bzw. mit Schlüsselwort NOTATION, falls die Elemente der Liste in der DTD definierte Notationen sind, z.B.: `NOTATION (DOC|BMP|GIF)`

Token-Typen

- ID:
  - Attribut muss in jedem Element einen (im gesamten XML-Dokument) eindeutigen Wert haben
- IDREF bzw. IDREFS:
  - Referenz bzw. Liste von Referenzen auf IDs
  - Listentrennung durch Zwischenräume
- ENTITY bzw. ENTITIES:
  - externe, nicht geparste Entität (s.u.), die in der DTD deklariert sein muss, bzw. Liste von solchen Entitäten
- NMTOKEN, NMTOKENS:
  - einzelner Name (entspricht erlaubtem XML Elementnamen) bzw. Liste von Namen

Beispiel: ID, IDREF

- ID/IDREF-Attribute in DTD deklarieren:

```
<!ELEMENT PRODUKT (#PCDATA)>
<!ATTLIST PRODUKT
       WarenCode ID #REQUIRED
       GehoertZu IDREF #IMPLIED>
```

- ID/IDREF-Attribute verwenden:

```
<PRODUKT WarenCode="S034">Kinderfahrrad</PRODUKT>
<PRODUKT WarenCode="S039" GehoertZu="S034">
    Gepäckträger
</PRODUKT>
```

Vorgabedeklarationen

`#REQUIRED:` Dieses Attribut muss im Dokument vorkommen

`#IMPLIED:` Dieses Attribut kann im Dokument vorkommen

`"wert":` Dieser Wert wird als Defaultwert angenommen, wenn kein Wert angegeben ist

`#FIXED "wert":` der Attributwert im Dokument ist der, der in der DTD erklärt wird (egal ob angegeben!)

Namespaces in DTDs

- in DTDs nur implizit: wie normale Attribute
- Namespace in Element- und Attributdeklaration angeben
- Beispiel:

```
<!ELEMENT stud:bewertung (#PCDATA)>
<!ATTLIST lehre xmlns:stud CDATA #REQUIRED>
```

- Namespace als fixes Attribut angebbar, um es nicht in jeder Dokumentinstanz wiederholen zu müssen

```
<!ATTLIST lehre xmlns CDATA #FIXED
  "http://www.lehre.tuwien.ac.at/lehre/2.0">
```

Beispiel 3

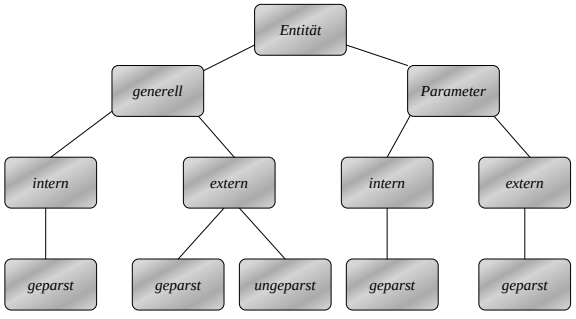
Entitäten in DTDs

- Entitäten:
  - Zeichenkette, die in der DTD als interne Entität definiert ist
  - externe Datei, die in der DTD als externe Entität definiert ist
  - Vordefinierte Entitäten
  - Character Entities
- Bei großen Projekten: zerlegen in kleinere Teilabschnitte mittels externer Entitäten
- DTD Auswertung: zunächst Entitätenexpansion; das veränderte Dokument und die DTD werden dann auf Wohlgeformtheit bzw. Gültigkeit geprüft

Arten von Entitäten in DTDs

- Generelle vs. Parameter Entität
  - Generell: innerhalb des Dokumentelements verwenden
  - Parameter Entität: innerhalb der DTD selbst verwenden **in interner DTD:** darf nicht in Elementen, Attributen, ... verwendet werden sondern **nur auf oberster Ebene**
- Interne vs. Externe Entität
  - Intern: Zeichenkette in Anführungszeichen
  - Extern: in einer separaten Datei
- Geparste vs. Ungeparste Entität
  - Geparst: XML-Text (Zeichendaten, Markup, ...). XML-Parser interpretiert den Inhalt
  - Ungeparst: beliebiger Datentyp (i.a. XML-fremde Daten)

Erlaubte Entitätstypen in DTDs:



Generelle, interne, geparste Entität

- In DTD deklarieren:

```
<!DOCTYPE ARTIKEL
  [<!ELEMENT ARTIKEL (TITELSEITE, EINLEITUNG, ABSCHNITT*)>
   <!ELEMENT TITELSEITE (#PCDATA|UNTERTITEL)*>
   ...
   <!ENTITY titel "Die Geschichte von XML
      <UNTERTITEL>Die Sprache des Web</UNTERTITEL>">
```

- In Dokumentelement referenzieren:

```
<ARTIKEL>
   <TITELSEITE>
      Titel: &titel;
   </TITELSEITE>
   ...
</ARTIKEL>
```

Generelle, externe, geparste Entität

- In DTD deklarieren:

```
<!DOCTYPE ARTIKEL ...
   <!ENTITY titel SYSTEM "Titel.dtd">
```

- Inhalt der Datei Titel.dtd

```
Die Geschichte von XML
<UNTERTITEL>Die Sprache des Web</UNTERTITEL>
```

- In Dokumentelement referenzieren:

```
<ARTIKEL>
   <TITELSEITE>
      Titel: &titel;
</TITELSEITE>
   ...
</ARTIKEL>
```

Generelle, externe, ungeparste Entität

- In DTD deklarieren:

```
<!DOCTYPE BUCH
  [<!ELEMENT BUCH (TITEL, AUTOR, COVERBILD)>
   ...
   <!ATTLIST COVERBILD Quelle ENTITY #REQUIRED>
   <!NOTATION GIF SYSTEM "ShowGif.exe">
   <!ENTITY christo SYSTEM "Christo.gif" NDATA GIF>
```

- In Dokumentelement referenzieren:

```
<BUCH>
   <TITEL>Der Graf von Monte Christo</TITEL>
   <AUTOR>Alexandre Dumas</AUTOR>
   <COVERBILD Quelle= "christo" />
</BUCH>
```

interne, geparste Parameterentität

- In DTD deklarieren und referenzieren:

```
<!ELEMENT lehre (veranstaltung+)>
<!ELEMENT veranstaltung (titel, schlagwort*, ...)>
...
<!ENTITY % boolean "(yes|no) 'no'">
<!ATTLIST zeit sine_tempore %boolean;>
```

Verwendung der Parameterentität in Element-, Attribut-, ... Definition: nur in externer DTD (oder in externer, geparster Parameterentität) erlaubt!

externe, geparste Parameterentität

- In DTD deklarieren und referenzieren:

```
<!DOCTYPE  BESTAND
  [<!ELEMENT BESTAND (BUCH | CD)*>
   <!ENTITY % buch_def SYSTEM "Buch.dtd">
   <!ENTITY % cd_def   SYSTEM "CD.dtd">
   %buch_def;
   %cd_def;
   ]
```

- Inhalt der Datei Buch.dtd:

```
<!ELEMENT BUCH (TITEL, AUTOR, COVERBILD)>
<!ELEMENT TITEL (#PCDATA | UNTERTITEL)*>
   ...
```

Weitere Entitäten

- Vordefinierte Entitäten
  - `<`
  - `>`
  - `&`
  - `&quot;`
  - `&apos;`
- Charakter-Entitäten (Unicode Wert)
  - `&#211;` (dezimal)
  - `&#xF3;` (hex)

Notationen

- Notation beschreibt ein bestimmtes Datenformat
  - URI eines Programms für Bearbeitung dieses Formats
  - URI eines Online-Dokuments, das dieses Format beschreibt
  - Einfache Beschreibung des Formats
- Verwendung:
  - Format einer generellen, externen, ungeparsten Entität beschreiben
  - um Attribut vom Aufzählungstyp NOTATION zu definieren
- Beispiele von Deklarationen in der DTD:

```
<!NOTATION DOC SYSTEM "WinWord.exe">
<!NOTATION BMP SYSTEM
     "http://www.dbai.tuwien.at/hilfe/bmp.html">
 <!NOTATION GIF SYSTEM "Graphic Interchange Format">
```

Konditionale Abschnitte

IGNORE / INCLUDE-Blöcke:

- Nur in externer DTD-Teilmenge bzw. in externer Parameter-Entität erlaubt
- IGNORE: „Auskommentieren“ von Teilen einer DTD

```
<![IGNORE[ <!ELEMENT name (fname, lname)> ]]>
```

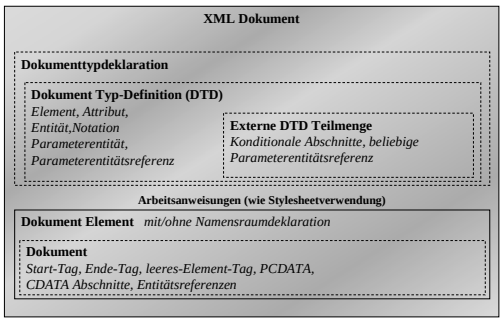
- INCLUDE: „Auskommentieren“ (vorübergehend) beseitigen

```
<![INCLUDE[ <!ELEMENT name (fname, lname)> ]]>
```

- Parameter-Referenz ist manchmal praktisch:

```
<!ENTITY % flexibel 'IGNORE'>
<![%flexibel;[ <!ELEMENT name (fname, lname)> ]]>
```

XML Dokument mit DTD



Nachteile von DTDs

- nicht in XML Syntax
- Rudimentäre Typisierung; beschränkte Auswahl an Attributtypen
- Keine Unterstützung von Namespaces
- Codierung beliebiger Kardinalitäten ist aufwendig
- keine objektorientierten Konzepte wie Vererbung; keine Wiederverwendbarkeit
- Rudimentäre Referenzierungen
- Erweiterbarkeit/Skalierbarkeit problematisch

⇨ *XML-Schema*

- Xerces:
  - Teil des Apache XML-Projekts
  - DOM- oder SAX-Parser
  - set CLASSPATH=xercesImpl.jar

Beispiel 4

- IE6:
  - XML als „Dateninsel“ in HTML-Seite
  - Ausgabe des XML-Inhalts in HTML-Seite
  - Gültigkeitsprüfung mit MSXML 4.0

Kurzbeschreibung: DOM vs. SAX

- Document Object Model (DOM)
  - DOM ist ein programmiersprachenneutrales Objektmodell plus Anwendungsprogrammierschnittstelle
  - beschreibt die Elemente, Attribute, Kommentare, Textteile und PIs eines XML-Dokuments als Objekte für die Verarbeitung mit einer objekt-orientierten Programmiersprache wie z.B. Java.
  - DOM liefert eine komplette Baumstruktur aller Objekte eines XML-Dokuments
  - eignet sich nicht für extrem große XML-Files. Gut für Editoren.
- Simple API for XML (SAX)
  - Programm-Schnittstelle für die Verarbeitung von XML-Dokumenten mit Hilfe einer objekt-orientierten Programmiersprache wie z.B. Java.
  - SAX liefert in einer eigenen Struktur Ereignisse in einem Eingabestrom. Eignet sich daher auch für sehr große XML-Files. Gut für App2App Austausch.

XML als „Dateninsel“ in HTML

- Verknüpfung des XML-Dokuments mit der HTML-Seite
  => Vorteile beider Umgebungen nützen:
  - XML: Strukturierung der Daten
  - HTML: Formatierungsfunktionen, Javascript
- Mittels HTML-Element „XML“ im BODY der HTML-Seite:

```
<XML
   ID = "dsoBuchbestand"
   SRC= "Buchbestand.xml">
</XML>
```

- Referenzierung mittels DATASRC-Attribut, z.B.:

```
<TABLE DATASRC = "#dsoBuchbestand">
```

Ausgabe des XML Inhalts

- Tabellendatenbindung (bei einfachen „Datensätzen“):
  - 1 Wurzelelement
  - alle „Datensatzelemente“ haben die gleichen Felder
  - Jedes Feldelement hat ausschließlich Zeichendaten.
- Verschachtelte Tabellen (bei hierarchischen Datensätzen)
- Weitere Möglichkeiten:
  - Datenbindung einzelner Datensätze
  - Scripts (DSO, DOM)

Vorgangsweise des IE6

- Beim Öffnen der HTML-Seite:
  - Analyse des XML-Dokuments durch MSXML-Parser
  - DOM-Parser
  - Aufbau des Programmierobjekts „DSO“ (Data Source Object)
  - Einfache Datensätze: als „record set“ gespeichert
  - DSO mit verschiedenen Methoden für Script-Programmierung
- Bei Fehler (Wohlgeformtheit, Gültigkeit):
  - IE6 zeigt XML-Daten einfach nicht an
  - in #dso...XMLDocument.parseError steht Fehler-Info

Ausgewählte Literatur/Webseiten

- http://www.oasis-open.org
  - XML Übersichtsseite
- http://www.xfront.com
  - XML Technologien Tutorials und Infos
- http://www.schemavalid.com/faq/xml-schema.html
  - XML Schema FAQ
- http://xml.apache.org
  - u.a. Xerces - validierender Parser
- Data Guides: Enabling Query Formulation and Optimization in semistructured data bases
  - R. Goldman und J. Widom, VLDB 1997
- Java Spektrum
  - deutschsprachige Java Zeitschrift mit XML Spektrum

Übungsbeispiel A

- Erstellen Sie für das XML Dokument laut Übungsbeispiel A des ersten Termins (d.h.: LVA-Homepage) eine DTD, die aus internem und externem Teil besteht.
- Überlegen Sie sich sinnvolle Varianten und berücksichtigen Sie diese in der DTD (z.B.: Auswahl mit „|“ oder optionale Attribute, ...)
- Versuchen Sie, neben Element- und Attribut-Deklarationen mindestens 5 weitere DTD-Sprachmittel zu verwenden, z.B.: verschiedene Arten von Entitäten, austauschbarer konditionaler Abschnitt, etc.
- Adaptieren Sie das XML-Dokument an diese DTD (z.B.: Referenzierung von Entitäten).
- Überprüfen Sie die Gültigkeit des XML Dokuments.

Übungsbeispiel B

- Erstellen Sie eine DTD für das XML Dokument laut Übungsbeispiel B des ersten Termins (d.h.: relationale Uni-Datenbank).
- Realisieren Sie die Primärschlüssel/Fremdschlüssel mittels ID/IDREF-Attributen.
- Definieren Sie unterschiedliche Namespaces für die 8 „Tabellen“. Verwenden Sie für einen dieser Namespaces den default-Namespace.
- Adaptieren Sie das XML-Dokument an diese DTD (z.B.: Namespaces, ID, IDREF berücksichtigen).
- Überprüfen Sie die Gültigkeit des XML Dokuments.