

XML-Praxis

# XML-Grammatiken

Jörn Clausen

`joern@TechFak.Uni-Bielefeld.DE`

# Übersicht

- formale Beschreibung von XML-Sprachen
- verschiedene Lösungen
- Document Type Definition
- Relax NG

# wohlgeformtes vs. valides XML

- *well formed XML*: bestimmte Kriterien erfüllt
  - korrekte Kodierung
  - korrekte Schachtelung der Elemente
  - korrekte Attribute
  - definierte Entitäten
- XML-Parser kann Wohlgeformtheit überprüfen
- *valid XML*: Dokument entspricht einer Grammatik

# Wozu eine Grammatik?

- formale Beschreibung der Sprache:
  - Welche Elemente gibt es?
  - Wie können sie kombiniert werden?
  - Welches Element besitzt welche Attribute?
  - Welche Attribut-Werte sind zulässig?
  - ...
- „Vertrag“ zwischen Erzeuger und Konsument von XML
- validierender Parser kann Konformität überprüfen
- Beispiele: (X)HTML, DocBook, SVG, MathML, SMIL, ...

# Qual der Wahl

- verschiedene Grammatik-Sprachen:
  - DTD** Document Type Definition, von SGML übernommen
  - W3C XML Schema** komplexes Typsystem
  - Relax NG** kompakte Notation, TREX von James Clark
  - Schematron** musterbasierte Beschreibung
  - DSDL** Document Schema Definition Language, ISO/IEC 19757
- Gewinner des Rennens?
- derzeit kontroverse Diskussionen

# Document Type Definition

- für allgemeine Daten nicht immer ideal
- für textuelle Daten aber (immer noch) geeignet
- leicht zu erlernen
- große Anzahl an etablierten DTDs
- große Anzahl an Werkzeugen (XML/SGML-Editoren)
- PSGML-Mode für Emacs

# Elemente definieren

- Element besitzt *content model*

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

- #PCDATA: Text ohne weitere Elemente
- Container-Elemente

```
<!ELEMENT slide (title, ilist)>  
<!ELEMENT ilist (item)*>
```

- Quantoren: ?, +, \* (wie in regulären Ausdrücken)
- Konnektoren:

a , b erst a, dann b

a | b a oder b

# Elemente definieren, cont.

- Mischung von Text und *inline*-Elementen

```
<item>... can be <emph>well formed</emph>  
or even ...</item>
```

- *mixed content*

```
<!ELEMENT item          (#PCDATA | emph)*>
```

- kann bei der Verarbeitung problematisch sein



# Aufgaben

- Die Datei `gedicht1a.xml` enthält ein kleines Gedicht. Schreibe eine passende DTD. Überprüfe mit Hilfe von `xmllint`, ob DTD und Gedicht passen:

```
$ xmllint --dtdvalid poem.dtd gedicht1a.xml
```

- Mit Hilfe des Elements „`index`“ sollen beliebige Wörter im Gedicht markiert werden, um später in einen Index aufgenommen zu werden:

```
<line>Es ist der <index>Vater</index>. Es ist ...</line>
```

Was ist zu tun, um Wörter im Titel, den Autorennamen oder Teile des eigentlichen Gedichts indexieren zu können?

# Attribute definieren

- „Aufzählungstyp“

```
<!ATTLIST presentation
    status (draft|final|publ) #REQUIRED>
```

- Vorgabe definieren

```
<!ATTLIST presentation
    status (draft|final|publ) "draft">
```

- beliebiger Text

```
<!ATTLIST presentation
    status (draft|final|publ) #REQUIRED
    date CDATA #IMPLIED>
```

# Aufgaben

- Definiere zwei Attribute für das Element „poem“ :
  - Mit „comment“ soll ein Freitext-Kommentar zum Gedicht angegeben werden können.
  - Gedichte müssen nun auch von der Freiwilligen Selbstkontrolle geprüft werden. Definiere ein verpflichtendes Attribut „fsk“, das die Werte „0“, „6“, „12“, „16“ und „18“ annehmen kann.
- Lege eine Kopie `gedicht1b.xml` der Datei `gedicht1a.xml` an. Füge in der Kopie die beiden Attribute ein.

# Entitäten definieren

- textuelle Makros

```
<!ENTITY xml "Extensible Markup Language">
```

- Umlaute wie in HTML

```
<!ENTITY Auml "&#196;">
```

```
<!ENTITY auml "&#228;">
```

- *parameter entity references*, für Makros in der DTD

```
<!ENTITY % text "(#PCDATA|ital|bold)*">
```

```
<!ELEMENT item %text;>
```

```
<!ELEMENT ital %text;>
```

```
<!ELEMENT bold %text;>
```

# Aufgaben

- Um das `index`-Element aus der vorletzten Aufgabe in alle gewünschten Inhaltsmodelle einzufügen, mußte an mehreren Stellen identischer Code verwendet werden. Verbessere die DTD mit Hilfe einer Parameter-Entität.

# DTD einbinden

- Verweis auf DTD im XML-Dokument

- *system identifier*

```
<?xml version="1.0"?>  
<!DOCTYPE presentation SYSTEM "presentation.dtd">
```

- *public identifier* und *system identifier*

```
<?xml version="1.0"?>  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

- PSGML-Mode des Emacs kann DTD lesen

# Aufgaben

- Füge eine DOCTYPE-Anweisung in `gedicht1b.xml` ein. Überprüfe nochmals die Validität der XML-Datei mit

```
$ xmllint --valid gedicht1b.xml
```

- Beginne eine neue XML-Datei im Emacs:

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<!DOCTYPE poem SYSTEM "poem.dtd">
```

und füge mit Hilfe des PSGML-Modes Elemente ein.

- Kopiere die Datei `gedicht2a.txt` oder `gedicht2b.txt` zu `gedicht2a.xml` bzw. `gedicht2b.xml`. Füge die XML-Deklaration und die DOCTYPE-Zeile ein und ergänze die Tags mit Hilfe des PSGML-Modes.

# Relax NG

- Relax Next Generation, sprich: *relaxing*
- <http://www.relaxng.org>
- entstanden aus
  - RELAX (Regular Language Description for XML), Murata Makoto
  - TREX (Tree Regular Expressions for XML), James Clark
- XML-Notation
- kompakte Notation, ähnlich zu EBNF



# Elemente definieren

- Äquivalent zu #PCDATA-Element:

```
<element name="title">  
  <text/>  
</element>
```

- Inhaltsmodell festlegen:

```
<element name="presentation">  
  <element name="title">  
    <text/>  
  </element>  
  <element name="author">  
    <text/>  
  </element>  
</element>
```

# Elemente definieren, cont.

- Wiederholungen:

```
<element name="ilist">  
  <oneOrMore>  
    <element name="item">  
      <text/>  
    </element>  
  </oneOrMore>  
</element>
```

- analog:

```
<zeroOrMore>...</zeroOrMore>  
<optional>...</optional>
```

# Aufgaben

- Die Datei `poem.rng` enthält den Rumpf einer Relax NG-Datei:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <start>

  </start>
</grammar>
```

- Füge innerhalb des `start`-Elements Definitionen ein, um `gedicht1a.xml` zu beschreiben. Überprüfe die Korrektheit der Grammatik mit

```
$ xmllint --relaxng poem.rng gedicht1a.xml
```

# Definitionen

- außerhalb des `start`-Elements einfügen:

```
<define name="itemList">
  <element name="iList">
    ...
  </element>
</define>
```

- Verwendung:

```
<element name="slide">
  <element name="title">
    <text/>
  </element>
  <ref name="itemList"/>
</element>
```

# Aufgaben

- Beschreibe das Element `verse` in einer eigenen Definition und verwende diese innerhalb von `poem`.

# Attribute definieren

- Attribute einem Element zuordnen:

```
<element name="presentation">  
  <attribute name="date">  
    <text/>  
  </attribute>  
  ...  
</element>
```

- optionales Attribut:

```
<element name="presentation">  
  <optional>  
    <attribute name="date">  
      <text/>  
    </attribute>  
  </optional>
```

# Attribute definieren, cont.

- Aufzählungstyp:

```
<attribute name="toc">  
  <choice>  
    <value>yes</value>  
    <value>no</value>  
  </choice>  
</attribute>
```

- funktioniert auch mit Elementen:

```
<element name="color">  
  <choice>  
    <value>black</value>  
    <value>white</value>  
  </choice>  
</element>
```

# Aufgaben

- Füge die Attribute `comment` und `fsk` dem Relax NG-Schema hinzu. Kontrolliere das Schema anhand der Datei `gedicht1b.xml`.



# Gruppen

- Liste von Koordinaten:

```
<points>  
  <x>0</x> <y>0</y> <z>0</z>  
  <x>1</x> <y>0</y> <z>0</z>  
</points>
```

- Definition:

```
<element name="points">  
  <oneOrMore>  
    <group>  
      <element name="x"> ...  
      <element name="y"> ...  
      <element name="z"> ...  
    </group>  
  </oneOrMore>  
</element>
```

# Alternativen

- Datumsformat:

```
<element name="date">
  <choice>
    <group>
      <element name="day"> ... </element>
      <element name="month"> ... </element>
      <element name="year"> ... </element>
    </group>
    <text/>
  </choice>
</element>
```

# Alternativen, cont.

- entweder

```
<date>  
  <day>24</day>  
  <month>Dezember</month>  
  <year>2003</year>  
</date>
```

oder

```
<date>24. Dezember 2003</date>
```

# mehr Flexibilität

- Attribute (fast) gleichberechtigt zu Elementen:

```
<element name="card">
  <choice>
    <element name="name"><text/></element>
    <attribute name="name"><text/></attribute>
  </choice>
  <choice>
    <element name="email"><text/></element>
    <attribute name="email"><text/></attribute>
  </choice>
</element>
```

# mehr Flexibilität, cont.

- erlaubte Instanzen:

```
<card>  
  <name>Joe User</name>  
  <email>juser@TechFak.Uni-Bielefeld.DE</email>  
</card>
```

```
<card name="Joe User">  
  <email>juser@TechFak.Uni-Bielefeld.DE</email>  
</card>
```

```
<card name="Joe User" email="juser@TechFak.Uni-Bielefeld.DE" />
```

- nicht erlaubt:

```
<card name="Joe User" email="juser@TechFak.Uni-Bielefeld.DE" />  
  <name>Joe User</name>  
</card>
```

# Aufgaben

- Ändere die Grammatik so ab, daß Titel und Autor des Gedichts gemeinsam entweder als Elemente oder als Attribute angegeben werden müssen:

```
<poem>
  <title>Der König Erl</title>
  <author>Heinz Erhardt</author>
  ...
</poem>
```

oder

```
<poem title="Der König Erl" author="Heinz Erhardt">
  ...
</poem>
```

# weitere Eigenschaften

- nicht gezeigt:
  - Datentypen, Einbindung von XSD-Datentypen
  - Listen
  - interleaving
  - Modularisierung
- Relax NG-Grammatiken, die mit DTD nicht möglich sind
- keine Entitäten

# kompakte Syntax

- XML-Notation *sehr* ausführlich
- Vorteil: kann mit XML-Werkzeugen gelesen/erzeugt werden
- Nachteil: unübersichtlich
- kompakte Notation: ähnlich zu DTD und EBNF
- verlustfreie Umformung zwischen beiden Darstellungen



# presentation.rnc

```
start = element presentation {
  attribute status { "draft" | "final" | "publ" },
  attribute date { text },
  element title { text },
  element author { text },
  element slide {
    element title {
      attribute toc { "yes" | "no" },
      text
    },
    itemList
  }+
}
itemList = element ilist {
  element item {
    (text
     | element emph { text }
     | element url { text })*
  }+
}
```

# Aufgaben

- Konvertiere `poem.rng` in die kompakte Syntax:  
`$ trang poem.rng poem.rnc`
- Konvertiere die Gedicht-DTD in ein Relax NG-Schema:  
`$ trang poem.dtd poem_from_dtd.rng`
- Erzeuge die Grammatik aus der Gedicht-Instanz:  
`$ trang gedicht1b.xml poem_from_xml.rng`
- Erzeuge ein W3C XML Schema:  
`$ trang poem.rng poem.xsd`
- Sieh Dir die entstandenen Dateien an und vergleiche sie.