Schema: Document Type Definition (DTD)

Markus Stocker

9. April 2018

Klausur

- 10. Juli 2018, 10-12 Uhr, schriftlich
- Zusammengelegt mit BIM-108-02 (Inhaltserschliessung I - Methoden)

Rekapitulation

- XPath, erklären Sie
  - Schritt und Lokalisierungspfad: Was bedeutet `A::KT[P]*`
  - Welche Lokalisierungspfadarten gibt es?
  - Was sind Prädikate?
  - Was erlauben Funktionen?
- Hatten wir bereits: Was bedeutet Wohlgeformtheit?

Übersicht

- Wozu ein Schema?
- Gültig: Nicht nur Wohlgeformt
- Die Document Type Definition (DTD)
- DTD Konstrukte und Beispiele

Schema: Warum brauchen wir sowas?

- Die NASA, ESA, und JAXA haben Daten über das Sonnensystem
- Man möchte nun die jeweiligen Daten austauschen und integrieren

Schema: Warum brauchen wir sowas?

- Die jeweiligen Systeme stellen Daten als XML zur verfügung
- Allerdings verwenden diese unterschiedliche *tags*
- Zum Beispiel `<planet/>`, `<Planet/>`, `<惑星/>`

Schema: Warum brauchen wir sowas?

- Zudem sind die XML Dokumente unterschiedlich strukturiert
- Solche Unterschiede erschweren den Datenaustausch
- Die Varianten in Software zu berücksichtigen ist mühsam
- Nicht nur mühsam, letztlich unmöglich da beliebig viele

Schema: Warum brauchen wir sowas?

- Was tun?
- Die drei Institutionen könnten sich auf ein Schema einigen

Schema: Warum brauchen wir sowas?

- Hier in diesem Raum legen wir uns auf die deutsche Sprache fest
- Würde ich die Vorlesung auf italienisch halten wäre das eher hinderlich
- Bei XML, der verwendeten Terme und deren Bedeutung ist es ähnlich
- Mit einem Schema legen sich Parteien auf einen *Standart* fest
- Dies soll die Interoperabilität der jeweiligen Systeme ermöglichen
- Fähigkeit eines Systems mit anderen Systemen zusammenzuarbeiten

Wohlgeformtheit und Gültigkeit

- Ein XML Dokument mit korrekter Syntax ist wohlgeformt
- Validiert ein wohlgeformtes Dokument einem Schema ist es gültig
- Ein gültiges Dokument ist immer auch wohlgeformt (*well-formed*)
- Ein wohlgeformtes Dokument ist nicht zwingend gültig (*valid*)

Document Type Definition (DTD)

- Eine Grammatik zur Spezifikation von XML Dokumente
- Beschreibung gültiger Elemente und Attribute
- Eine DTD definiert somit auch die XML Struktur
- Parteien können sich auf eine gemeinsame DTD einigen
- Systeme werden dann entsprechend der DTD entwickelt
- Systeme können XML Dokumente der DTD entgegen validieren
- Erhaltene aber auch gelieferte XML Dokumente
- Validiert ein XML Dokument ist dieses ...

Beispiel: XML Dokument und Entsprechende DTD

```
<planets>
  <planet>
    <name>Earth</name>
  </planet>
</planets>
```

```
<!ELEMENT planets (planet)>
<!ELEMENT planet (name)>
<!ELEMENT name (#PCDATA)>
```

Einbindung einer DTD: Im Dokument

```
<?xml version="1.0"?>
<!DOCTYPE planets [
<!ELEMENT planets (planet)>
<!ELEMENT planet (name)>
<!ELEMENT name (#PCDATA)>
]>
<planets>
  <planet>
    <name>Earth</name>
  </planet>
</planets>
```

Einbindung einer DTD: Extern

```
<?xml version="1.0"?>
<!DOCTYPE planets SYSTEM "planets.dtd">
<planets>
  <planet>
    <name>Earth</name>
  </planet>
</planets>
```

Inhalt der `planets.dtd` Datei

```
<!ELEMENT planets (planet)>
<!ELEMENT planet (name)>
<!ELEMENT name (#PCDATA)>
```

Konstrukte: Deklaration deren Verwendung

- XML Dokumente verwenden folgende Konstrukte
  - Elemente
  - Attribute
  - Entitäten
  - PCDATA (*parsed character data*)
  - CDATA (*[unparsed] character data*)
- Wie wir gesehen haben, allerdings sehr unterschiedlich
- DTD ermöglicht Deklaration spezifischer Verwendungen
- Also, Deklaration gültiger Verwendung der Konstrukte

Elemente Deklarieren

- Elemente werden mit `ELEMENT` deklariert
- Entsprechend der folgenden Syntax

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

- Der `name` des elements
- Und `inhalt` einer von
  - `EMPTY` für das Element ohne Inhalt
  - `ANY` für beliebige Inhalte
  - Gemischte Inhalte, `PCDATA` und Kindelemente
  - Kindelemente

```
<!ELEMENT planets EMPTY>
<!ELEMENT planet (name, radius)>
<!ELEMENT name (#PCDATA)>
```

Element Inhaltsmodelle

- Sequenz `(A, B)`: `A` und `B` müssen in dieser Sequenz auftreten
- Alternative `(A | B)`: Entweder `A` oder `B` müssen auftreten
- Wiederholungen
  - `*`: Null oder mehrmals
  - `+`: Ein oder mehrmals
  - `?`: Null oder einmal

```
<!ELEMENT planets (planet+)>
<!ELEMENT planet (name, radius?)>
<!ELEMENT planet (#PCDATA | name)*>
```

Attribute Deklarieren

- Attribute werden mit `ATTLIST` deklariert
- Entsprechend der folgenden Syntax

```
<!ATTLIST element attribut typ wert>
```

- Der `element` Name
- Der `attribut` Name
- Der `typ` des Attributs
- Der `wert` des Attributs

```
<!ATTLIST name radius CDATA "0">
```

Attribut Typen (ausgewählte)

Typ	Beschreibung
`CDATA`	Wert ist Zeichenfolge
`(w1|w2|...)`	Wert aus enumerierter Liste
`ID`	Wert ist eindeutige ID
`IDREF`	Wert ist ID eines anderen Elements
`NMTOKEN`	Wert ist ein gültiger XML Name
`ENTITY`	Wert ist eine Entität

Attribut Werte

Wert	Beschreibung
`value`	Der Standartwert des Attributs
`#REQUIRED`	Das Attribut ist zwingend
`#IMPLIED`	Das Attribut ist optional
`#FIXED value`	Das Attribut ist optional, Wert ist festgelegt

Entitäten Deklarieren

- Entitäten werden mit `ENTITY` deklariert
- Entsprechend der folgenden Syntax

```
<!ENTITY name "wert">
```

- Der `name` der Entität
- Der `wert` der Entität

```
<!DOCTYPE planet [
  <!ENTITY earth "Planet Earth">
]>
<planet>
  <name>&earth;</name>
</planet>
```

Namensräume

- DTD unterstüzt Namensräume nicht
- Man kann diese allerdings als Attribute deklarieren

```
<!ELEMENT nasa:planet EMPTY>
<!ATTLIST nasa:planet xmlns:nasa
          CDATA #FIXED "http://nasa.org">

<nasa:planet xmlns:nasa="http://nasa.org"/>
```

Zusammenfassung

- Schemas ermöglichen und erhöhen Interoperabilität
- Fähigkeit verschiedener Systeme zusammenzuarbeiten
- Nötig ist eine Sprache zur Spezifikation von XML Dokumente
- DTD ist eine solche Sprache
- Mit DTD kann man XML Dokumente validieren
- Validiert ein XML Dokument ist dieses gültig
- Ein gültiges XML Dokument ist immer auch wohlgeformt