Schema: XML Schema

Markus Stocker

16. April 2018

Rekapitulation

- Wozu brachen wir Schemata?
- Was ist die Document Type Definition (DTD)?
- Was ist der Unterschied zwischen Wohlgeformtheit und Gültigkeit?
- Unterstützt DTD Namensräume?
- Gültig oder nicht? Wenn nicht, warum?

```
<!ELEMENT planets (planet)>
<!ELEMENT planet EMPTY>

<planets>
  <planet/>
  <planet/>
</planets>
```

Übersicht

- Warum DTD nicht genügt
- XML Schema
- Sprachkonstrukte
- Beispiele

Wir haben die DTD ...

- Warum brauchen wir eine weitere Sprache für Schema Definition?
- DTD hat ein paar Nachteile, und zwar
  - Keine Datentypen
  - Keine Unterstützung für Namensräume
  - Nicht XML Syntax
- XML Schema behebt diese Nachteile
- Der Zweck ist gleich: XML Dokumente auf Gültigkeit prüfen

XML Schema

- Eine XML basierte Alternative zu DTD
- XML Schema ist also in XML geschrieben
- Nicht nötig eine weitere Sprache zu lernen
- Gleichen Tools, wie Editor, Parser, etc.
- Unterstützt Datentypen und stellt vordefinierte zur Verfügung
- Ermöglich detailliertere Beschreibung der Daten
- Strengere Restriktionen die auf Gültigkeit geprüft werden können
- Einfachere Konvertierung zwischen Datentypen

XML Schema

- XML Schema ist erweiterbar, weil in XML geschrieben
- Ein Schema kann andere Schema verwenden und erweitern
- Eigene Datentypen definieren (Benutzerdefiniert)
- Diese werden aus den definierten Datentypen abgeleitet
- Ein XML Dokument kann auf mehrere XML Schema referenzieren

XML Schema

- Unterstützung von Datentypen ist wichtig
- In DTD sind die Zeichenketten `2018-02-10` und `0.5` `PCDATA`
- Genauer genommen, handelt es sich aber um ein Datum und eine Zahl
- Es ist von Vorteil wenn man dies beschreiben kann
- Ausserdem, ist das Datum `2018-02-10` nicht eindeutig
- Es könnte der 2. Oktober oder der 10. Februar 2018 sein
- Der Datentyp `xs:date` spezifiziert das Format als `YYYY-MM-DD`
- Somit ist klar, dass `2018-02-10` der 10. Februar 2018 ist

Beispiel

XML Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="name" type="xs:string"/>
</xs:schema>
```

DTD (zum Vergleich)

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

XML (als Beispiel)

```
<name>Earth</name>
```

Elemente Deklarieren

- Unterscheidung zwischen einfachen und komplexen Elemente
- Einfache Elemente sind XML Elemente die Text beinhalten
- XML Elemente ohne Kindelemente oder Attribute
- Wobei “Text” viel sein kann: Zahlen, Datum, Zeichenfolge, ...
- XML Elemente mit Kinder/Attribute sind komplexe Elemente

Einfache Elemente Deklarieren

```
<xs:element name="..." type="..."/>
```

- Der `name` entspricht dem Elementnamen
- Der `type` entspricht dem Elementtyp

Vordefinierte Datentypen (Auswahl)

- `xs:string`
- `xs:boolean`
- `xs:int`
- `xs:date`
- `xs:time`
- `xs:dateTime`
- `xs:duration`

Einfache Elemente: Beispiel

```
<xs:element name="name" type="xs:string"/>
<xs:element name="radius" type="xs:decimal"/>

<name>Earth</name>
<radius>6371.0</radius>
```

Einfache Elemente: *Default* und *Fixed* Werte

- Einfache Elemente können vorgegebene oder festgelegte Werte haben
- Der vorgegebene Wert wird dem Element automatisch zugewiesen
- Angenommen es wird kein anderer Wert angegeben
- Der festgelegte Wert wird ebenfalls automatisch zugewiesen
- Es ist nicht möglich ein anderer Wert anzugeben

```
<xs:element name="name" type="xs:string" fixed="Earth"/>
<xs:element name="radius" type="xs:decimal" default="0"/>
```

Einfache Elemente: *minOccurs* und *maxOccurs*

- Spezifikation der Häufigkeit eines Elements
- *minOccurs*: Wie oft ein Element mindestens vorkommen muss
- *maxOccurs*: Wie oft ein Element höchstens vorkommen kann
- Vorgegebener Wert ist 1
- *minOccurs* darf nicht grösser sein als *maxOccurs*
- Unlimitierte Häufigkeit mit `maxOccurs="unbounded"`

```
<xs:element name="name" type="xs:string" maxOccurs="3"/>
<xs:element name="radius" type="xs:decimal" minOccurs="0"/>
<xs:element name="name" type="xs:string" minOccurs="2" maxOccurs="10"/>
<xs:element name="name" type="xs:string" maxOccurs="unbounded"/>
```

Attribute Deklarieren

```
<xs:attribute name="..." type="..."/>
```

- Der `name` entspricht dem Attributnamen
- Der `type` entspricht dem Attributtyp
- Ähnlich wie die Deklaration einfacher Elemente
- Einfache Elemente können keine Attribute haben
- Attribute selbst werden allerdings als einfache Elemente deklariert

Attribute: Beispiel

```
<xs:attribute name="radius" type="xs:decimal"/>

<name radius="6371.0">Earth</name>
```

Attribute: Weitere Eigenschaften

- Attribute können vorgegebene oder festgelegte Werte haben
- Attribute sind optional: Mit `use="required"` sind sie erforderlich

Komplexe Elemente Deklarieren

- Deklaration von XML Elemente mit Kindelemente und/oder Attribute
- Es gibt vier Arten von komplexen Elemente
  - Leere Elemente
  - Elemente die nur andere Elemente enthalten
  - Elemente die nur Text enthalten
  - Elemente die Text und andere Elemente enthalten
- Können Attribute enthalten
- Folgendes Beispiel ist ein komplexes Element
- Der Inhalt ist zwar nur Text (einfaches Element)
- Allerdings enthält das Element ein Attribut

```
<name radius="6371.0">Earth</name>
```

Komplexe Elemente Deklarieren: Variante I

```
<xs:element name="planet">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="radius" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Komplexe Elemente Deklarieren: Variante II

```
<xs:element name="planet" type="PlanetType"/>

<xs:complexType name="PlanetType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="radius" type="xs:decimal"/>
  </xs:sequence>
</xs:complexType>
```

Komplexe Elemente Deklarieren: Vergleich

- Variante I entspricht dem XML Dokument und ist einfach
- In Variante I kann nur `planet` die Spezifikation verwenden
- In Variante II können mehrere Elemente die Spezifikation verwenden
- Andere Elemente mögen ein Name und Radius haben
- Spezifiziere den Typ einmal und verwende ihn mehrmals
- Weiterer Vorteil der Variante II ist die mögliche Erweiterung

Komplexe Elemente Erweitern

```
<xs:element name="ellipsoid" type="EllipsoidType"/>
<xs:element name="planet" type="PlanetType"/>

<xs:complexType name="EllipsoidType">
  <xs:sequence>
    <xs:element name="radius" type="xs:decimal"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="PlanetType">
  <xs:complexContent>
    <xs:extension base="EllipsoidType">
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Komplexe Leere Elemente

```
<planet name="Earth"/>

<xs:element name="planet">
  <xs:complexType>
    <xs:attribute name="name" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

Komplexe Elemente die nur Text Beinhalten

```
<planet radius="6371.0">Earth</planet>

<xs:element name="planet">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="radius" type="xs:decimal" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Komplexe Elemente Ordnen

- Es ist möglich, die Ordnung der Kindelement zu spezifizieren
- `xs:all`: Beliebige Ordnung und jedes Element nur einmal
- `xs:choice`: Ein Element aus der spezifizierten Menge
- `xs:sequence`: Elemente müssen in spezifizierter Ordnung vorkommen

Einschränkungen (*Restrictions*)

- XML Element und Attribut Werte können eingeschränkt werden
- Möglich sind Restriktionen auf
  - Wertebereich, z.B. `1000 < radius < 10000`
  - Wertemenge, z.B. `Farben = {Blau, Rot, Grün}`
  - Wertemuster, z.B. `[a-zA-Z]`
  - Leerzeichen, z.B. `collapse`
  - Länge, z.B. Passwort mit Länge 10 Zeichen

Einschränkungen: Wertebereich

```
<xs:element name="radius">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1000"/>
      <xs:maxInclusive value="10000"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Einschränkungen: Wertemenge

```
<xs:element name="planet">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Venus"/>
      <xs:enumeration value="Earth"/>
      <xs:enumeration value="Mars"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Einschränkungen: Wertemuster

```
<xs:element name="name">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][a-z]+"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Namesräume

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://example.org">

  <xs:element name="planet">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="radius" type="xs:decimal"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Zusammenfassung

- XML Schema behebt einige Nachteile der DTD, isb. Datentypen
- Gleicher Zweck, XML Dokumente auf Gültigkeit prüfen
- Grundidee ist benötigte Elemente und Attribute zu deklarieren
- XML Schema und DTD ermöglichen und erhöhen Interoperabilität