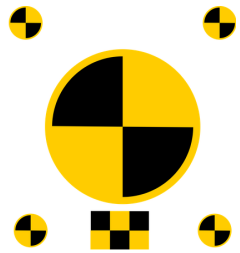
Einführung in XML Schema

Dr. Björn Rudzewitz

pagina GmbH

15.02.2023



pagina GmbH Publikationstechnologien -

Die Technologieexperten für das Publizieren im digitalen Zeitalter.

Nachhaltig.

Besuchen Sie unsere Webseite: https://pagina.gmbh/

Ihre Moderatoren des heutigen Abends

- Dr. Björn Rudzewitz, pagina GmbH (Folien)
- Dr. Heino Schmull, pagina GmbH (Chat, Praxisbeispiele)

Lernziele

Lernziele:

- ✓ einsteigerfreundliche Einführung in Grundlagen
- ✓ Vergleich mit Alternativen zu XML Schema
- ✓ Illustration durch praxisnahe Beispiele

Lernziele

Nicht Teil der Lernziele:

- X umfangreiche, fortgeschrittene Schemata
- X praktische Validierung in Werkzeugen
- X vollständige Beschreibung des Featureumfangs

Motivation

- XML: Meta-Markup-Sprache
- XML definiert die Wohlgeformtheit eines XML-Dokuments
- **aber**:
  - keine Annahmen über die Struktur eines Dokuments
  - Struktur muss formal dokumentiert und überprüfbar sein
  - Prozesse, die XML verarbeiten, verlassen sich auf eine bestimmte Struktur

Warum XML Schema?

- diverse Ansätze zur Überprüfung der Struktur von XML-Dokumenten:
  - DTD
  - RelaxNG
  - XML Schema/XML Schema Definition (XSD)
- Vorteile XML Schema:
  - modern und weit verbreitete Nutzung
  - definiert und geschrieben **in XML für XML** (um es zu beschreiben)
  - durch Datentypen mächtiger als DTD
  - Dokumentationsfelder → Online-Doku
  - beliebter als RelaxNG → Community

Warum XML Schema?

großer Vorteil von XML Schema:

- Schwerpunkt Datentypen
- Text nicht nur als Text
- stattdessen: Regeln, Typen (z.B. Datum, Bool'sche Werte, . . .)
- Möglichkeit zu eigenen Definition

Zweck von XML Schema

- formale Definition von Regeln zur Struktur von XML-Formaten
- maschinenlesbar
- menschenlesbar
- automatische Validierung
- Dokumentation von XML-Sprachen für Wiederverwendbarkeit

Referenzierung von XML Schemas

- Schema Referenzierung:
  - via Attribut `schemaLocation` im Namensraum `xsi` (Schema Instance)
  - Laufzeit-Argument für Werkzeuge
  - Konfigurationsdateien

Das XML Schema Element

- definiert als XML Dokumente
- Wurzelelement: `schema`
- *xs* Namensraum für alle vordefinierten Schemakomponenten

```
<?xml version="1.0" encoding="UTF-8" ?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- content of schema goes here -->
</xs:schema>
```

Inhalte modellieren

- Inhaltsmodell: wie XML-Komponenten modelliert/strukturiert werden
- einfache Typen: ausschließlich Textinhalt
  - großer Vorteil: Typisierung von Textinhalten
  - neben Textinhalt von Elementen auch Attributwerte
- komplexe Typen: immer dann wenn mehr als nur Text verwendet wird (z.B. Kindelemente, Attribute)

Definition von einfachen Elementen

- **einfaches Element**: enthält nur Text
- aber: Datentypen des Textes können in *type* Attributen definiert werden, z.B. *xs:string*, *xs:decimal*, *xs:boolean*, *xs:date*
- Standardwerte möglich

Definition von einfachen Elementen

Beispiele:

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

<xs:element name="impactfactor" type="xs:decimal"/>

<xs:element name="publicationdate" type="xs:date"/>

<xs:element name="license" type="xs:string"
  default="public domain"/>
```

Attribute deklarieren

- wie Elemente werden Attribute definiert mit bestimmten Typen und möglicherweise Standardwerten
- wenn nicht anders angegeben: Attribute sind optional, Verwendung von *use* um verpflichtend vorzuschreiben

Beispiele:

```
<xs:attribute name="numAuthors" type="xs:integer"/>
<xs:attribute name="lang" type="xs:string" default="EN"/>
<xs:attribute name="lang" type="xs:string" use="required"/>
```

Facetten

- **Facette**: Beschränkung des Textinhalts von Elementen/Attributen
- Definition von speziellen Werten von Text gegeben eine generische *type* Definition
- großer Vorteil von XML Schema gegenüber z.b. DTDs

Facette – Beispiel

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Quelle: https://www.w3schools.com/xml/schema_facets.asp

Facette

Längenbeschränkung:

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Facette – Beispiel

- globale Typen können für Elemente/Attribute wiederverwendet werden
- Beispiel: enumeration (nur bestimmte Werte erlaubt)

```
<!--Metadaten Barrierefreiheit-->
<xs:attribute name="kontrastreiche-anzeige" type="ja-nein"/>
<!--Kapitel-->
<xs:attribute name="toc" type="ja-nein"/>
<!--globales Attribut-->
<xs:attribute name="print" type="ja-nein"/>
<!--Deklaration:-->
<xs:simpleType name="ja-nein">
   <xs:restriction base="xs:token">
      <xs:enumeration value="ja"/>
      <xs:enumeration value="nein"/>
    </xs:restriction>
</xs:simpleType>
```

Facette

- XML Schema erlauben (limitierte) reguläre Ausrücke
- Zeichenklassen, Quantifikatoren, Wiederholungen, ...
- *xs:pattern* zur Definition

Facette – Beispiel

```
<ISBN>978-3-12345-456-X</ISBN>

<!-- Deklaration -->
<xs:element name="ISBN">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="978-\d-\d{3}-\d{5}-[\dX]"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Facette

Weitere Beispiele für Facetten:

- *fractionDigits*: Nachkommastellen
- *minLength*
- *totalDigits*
- *whiteSpace*: Behandlung nicht-druckbarer Zeichen (*replace*, *collapse*, *preserve*)
- *unique*: Eindeutigkeit (z.B. ISBN)

vollständige Referenz: https://www.w3schools.com/xml/schema_facets.asp

Lokale vs. Globale Definitionen

- *globale* Komponenten: direkt unter dem Wurzelelement `xs:schema` definiert
- *lokale* Komponente: definiert unter anderen Elementen

Vorteile von globalen Komponenten:

- einmal definieren und in diesem oder anderen Schemata referenzieren
  - Elemente und Attribute
  - Typen, mit denen dann Elemente/Attribute deklariert werden können
- Vermeidung von Redundanz indem das gleiche Element nur einmal lokal definiert wird statt mehrfach an verschiedenen Stellen
- Benennung von globalen Typen und Wiederverwendung im Schema
- Überschreibung globale Definition durch lokale wo nötig

ref Attribut

- `ref` Attribut: Referenzierung von Elementen und Attributen, auch aus anderen Schemata
- für die Benutzung: zuerst alle Elemente und Attribute definieren, dann Referenzierung via `ref`
- `<xs:group>`/`<xs:attributeGroup>`: häufig benötigte Gruppen global deklarieren und dann referenzieren: z.B. globale Attribute oder Elemente, die in verschiedenen Kontexten als Auswahl vorkommen: Schriftauszeichnungen, Überschriften, Listen

```
<xs:group name="schrift">
        <xs:choice>
                <xs:element ref="kursiv"/>
                <xs:element ref="code"/>
                <xs:element ref="durchgestr"/>
                <xs:element ref="fett"/>
                <!-- ... -->
        </xs:choice>
</xs:group>
```

ref Attribut

```
<xs:element name="firstname" type="xs:string"/>
<xs:element name="lastname" type="xs:string"/>

<xs:attribute name="userid" type="xs:integer"/>

<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="firstname"/>
      <xs:element ref="lastname"/>
    </xs:sequence>
    <xs:attribute ref="userid" use="required"/>
  </xs:complexType>
</xs:element>
```

Komplexe Elemente

Komplexe XML Elemente:

- leer
- mit Kindelementen
- mit Attributen
- mit gemischtem Inhalt

komplexe Elemente definiert durch *Indikatoren*

Komplexe Elemente – Elemente mit Kindelementen

- *sequence* Indikator zur Definition von Kindelementen (in dieser Reihenfolge)

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Quelle: `https://www.w3schools.com/xml/schema_complex.asp`

Komplexe Elemente – Elemente mit Kindelementen

- *all* Indikator zur Definition von Kindelementen (jede Reihenfolge, alle Elemente maximal einmal)
- *choice*: exklusive Wahl von Kindelementen (aber: mit *maxOccurs='unbounded'* auch mehrere Vorkommen)

Beispiel:

```
<xs:element name="urheber">
  <xs:complexType>
      <xs:all>
        <xs:element minOccurs="0" ref="autoren"/>
        <xs:element minOccurs="0" ref="co-autoren"/>
        <xs:element minOccurs="0" ref="herausgeber"/>
        <xs:element minOccurs="0" ref="uebersetzer"/>
        <xs:element minOccurs="0" ref="illustratoren"/>
      </xs:all>
  </xs:complexType>
</xs:element>
```

Komplexe Elemente – Elemente mit Kindelementen

- *minOccurs* und *maxOccurs* Indikatoren für Anzahl von Elementen

```
<xs:element name="uebersetzer">
  <xs:complexType>
    <xs:sequence>
      <!-- zuerst alle Autoren als <person> -->
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="per
      <!-- beschreibender Textabsatz am Ende -->
      <xs:group minOccurs="0" ref="urheber-abs"/>
      <!-- minOccurs=0: auch nur <person> oder nur Absatz -->
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Komplexe Elemente – leere Elemente

- Definition von leeren Elementen: komplexer Typ, aber ohne Textinhalt oder Kindelemente

Beispiel:

```
<!-- leeres Element mit Attributen -->
<xs:element name="cover">
    <xs:complexType>
        <xs:attribute name="quelle"/>
        <xs:attribute name="ersatztext"/>
    </xs:complexType>
</xs:element>
```

Komplexe Elemente – nur Text

- *simpleContent* Element
- verpflichtend: Erweiterung oder Beschränkung innerhalb von *simpleContent* Element

Beispiel: Ziffer (nur Text und Attribute, keine Kindelemente)

```
<xs:element name="ziffer">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attributeGroup ref="globale_attribute"/>
                <!-- @id, @sprache etc. -->
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
```

Komplexe Elemente: beliebige Elemente

- *any* um jedes Element zu erlauben
- im Gegensatz zu DTDs erlaubt *any* sogar Elemente, welche nicht im aktuellen Schema deklariert sind

```
<!-- Kindelement von <meta>: verlagsspezifische Daten -->
<!-- mit einem zusaetzlichen Schema definiert -->
<xs:element name="erweiterungen" minOccurs="0">
    <xs:complexType>
        <xs:choice>
            <xs:any minOccurs="0" maxOccurs="unbounded"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

Zusammenfassendes Beispiel aus der Praxis

→ absatz.xml

Referenzen

Folien und Beispiele zum Teil angelehnt an W3Schools Tutorial zu XML Schema (https://www.w3schools.com/xml/schema_intro.asp, zuletzt aufgerufen am 03.02.2023). Beispiele aus der Verlagsbranche abgeleitet vom XML-first Framework und Schema parsX - mehr Informationen auf der parsX-Webseite (https://www.parsx.de, zuletzt aufgerufen am 14.02.2023).

Die Folien wurden von Dr. Björn Rudzewitz für das Seminar *Text Technology* entwickelt und in der Lehre an der Universität Tübingen im SS 19/20/21 eingesetzt und für den Vortrag zusammen mit Dr. Heino Schmull übersetzt und überarbeitet.