

XML-Schema:

Matthias Bibo

19.06.2006

Ausarbeitung für das Seminar

Webtechnologie

unter der Leitung von
Thomas Gottron und
Prof. Dr. Herbert Göttler

Johannes Gutenberg Universität Mainz

Inhaltverzeichnis

1. Was ist XML-Schema?
 - 1.1 Vorteile von XML-Schema gegenüber DTD
2. Aufbau eines XML-Schemas
3. Datentypen
 - 3.1 Einfache Typen
 - 3.1.1 Atomare Typen
 - 3.1.2 Aggregierte Typen
 - 3.2 Komplexe Typen
 - 3.2.1 Kompositoren
 - 3.2.2 Komplexe Typen ableiten
 - 3.3 Anonyme Typen
 - 3.4 Häufigkeitsbeschränkungen
 - 3.5 Attribute
 - 3.6 Globale Elemente und Attribute
4. Wiederverwendbarkeit
5. Quellenangabe

1. Was ist XML-Schema?

Das Schema,

Mehrzahlformen: die **Schemas, Schemen** oder

Schemata. Bedeutet soviel wie Anschein (haben), Figur, Haltung, Muster oder Entwurf, welcher die groben Umrisse einer bestimmten Erscheinungsform wiedergibt.

XML-Schema wurde am 2.Mai 2001, nach einer Entwicklungszeit von etwa 3 Jahren, von der W3C verabschiedet. Es wird in folgende drei Teile gegliedert:

- XML Schema Teil 0: Einführung
- XML Schema Teil 1: Strukturen
- XML Schema Part 2: Datentypen

Ein XML-Schema hat im Großen und Ganzen den selben Zweck wie eine Dokumenttypdefinition. Es legt die Struktur eines XML-Dokumentes fest, indem es unter anderem vorgibt, welche Elemente in einem Dokument vorkommen dürfen, in welcher Reihenfolge die Elemente vorkommen müssen und welchen Typ die Elemente haben sollen. Wenn z.B. in einem XML-Dokument ein Element vorkommt, das nicht von dem Typ ist wie es im dazugehörigen XML-Schema definiert ist, ist das XML-Dokument nicht konform zum Schema und dadurch nicht korrekt.

Ein XML-Schema ist selbst ein XML-Dokument und lässt sich deshalb auch durch ein DTD oder ein XML-Schema beschreiben.

1.1 Vorteile von XML-Schema gegenüber DTD

DTD:

- haben einen stark eingegrenzten Typvorrat (PCDATA)
- unterstützen keine Namensräume
- sind nicht XML-konform

XML-Schema:

- ist ein XML-Dokument
- verfügt über objektorientierte Konzepte
- ist leicht erweiterbar
- unterstützt die Verwendung von Namensräumen
- umfangreiche Datenprüfungen
- hat den vordefinierten Datentyp „date“

2. Aufbau eines XML-Schemas

Beispiel:

```
<schema (Attribute) > ----> Wurzelement
  <annotation> .....</annotation> ----> Anmerkung
  <element name= "... " type= "..... " /> ----> Elementdeklaration
  <simpleType name= "... " > ----> Einfacher Typ
</simpleType>
  <complexType name= "... "> ----> Komplexer Typ
    <element name= "... " type= "..."/>
    <attribute name= "... " type= "..."/> ----> Attributdeklaration
  </complexType>
</schema>
```

Wurzelement: Im Wurzelement wird der vom Schema verwendete Namensraum angegeben und die globalen Typen deklariert.

Anmerkung: Bei Anmerkungen innerhalb eines XML-Schemas wird Grundsätzlich der Befehl "annotation" verwendet. Danach gibt es folgende zwei Befehle, die man verwenden kann:

appinfo: Um eine Anmerkung für Hilfsprogramme, Stylesheets und andere Anwendungen unterzubringen.

Documentation: Mit dem documentation-Attribut kann man Anmerkungen für den Anwender unterbringen.

Element- und Attributdeklaration : Bei der Element- bzw. Attributdeklaration wird der Name des Elements oder Attributs angegeben. Danach wird der Typ bestimmt, von dem das Element oder das Attribut sein soll.

Einfacher Typ: Die Deklaration eines einfachen Typs wird durch den Befehl "simpleType" eingeleitet. Danach folgt der Name und der Typ. Dies kann z.B. ein abgeleiteter Basistyp sein.

Komplexer Typ: "complexType" ist der Befehl, der die Deklaration eines komplexen Typs einleitet. Es folgt, wie auch beim einfachen Typ, der Name. Danach werden die Kindelemente und Attribute deklariert, die der komplexe Typ enthalten soll.

3. Datentypen

Zu den Datentypen in XML-Schema gehören die einfachen Typen, die komplexen Typen und die anonymen Typen

3.1 Einfache Datentypen

Die einfachen Datentypen setzen sich aus den atomaren Typen und den aggregierten Typen zusammen.

3.1.1 Atomare Typen

Zu den atomaren Typen gehören die Grundtypen, wie z.B. integer, string, double etc., wie man sie aus vielen Programmiersprachen kennt, und die abgeleiteten Typen. Diese können durch die Einschränkung von Grundtypen, mittels sogenannter "Fassetten" definiert werden.

Zu den Fassetten gehören z.B.:

minInclusive, maxInclusive:

Einschränkung eines Integers durch das Setzen eines minInclusiv und/ oder maxInclusiv- Wertes, die angeben welchen Wert mein Integer mindestens haben muss, bzw. Maximal haben kann.

enumeration:

Mit der enumeration kann man eine Liste von möglichen Werten angeben, die ein Typ haben kann.

length:

Gibt die genaue Länge eines Typs an, d.h. wieviele Stellen er haben muss.

fractionDigits, totalDigits

Mit dem fractionDigits Befehl kann man die Anzahl der Nachkommazahlen einer Zahl angeben, mit totalDigits die Anzahl der gesamten Stellen.

pattern:

Die Fassetten pattern legt mit Hilfe eines regulären Ausdrucks ein Muster an, wie der Datentyp aufgebaut sein muss.

Beispiel:

In diesem Fall wird eine String abgeleitet, indem mit Hilfe der Fassetten pattern das Muster des Typs vorgegeben wird. Der einfache Typ ISBNTyp müsste also aus 9 Ziffern und am Ende einem Zeichen, welches entweder den Wert 0 - 9 oder X hat, bestehen.

```
<xsd:simpleType name="ISBNTyp">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{9}[0-9X]"/>
  </xsd:restriction>
</xsd:simpleType>
```

3.1.2 Aggregierte Typen

Zu den aggregierten Typen zählen Listen und Vereinigungen.

Listen können immer nur einfache Typen enthalten, deshalb gibt es keine Listen von Listen oder von komplexen Typen. Die Einträge bei einer Liste werden immer durch Leerzeichen getrennt. Der Listentyp kann durch die Fassetten length, minLength, MaxLength und enumeration eingeschränkt werden.

XML-Schema stellt drei vorgefertigte Listentypen zur Verfügung: NMTOKENS, IDREFS und ENTITIES.

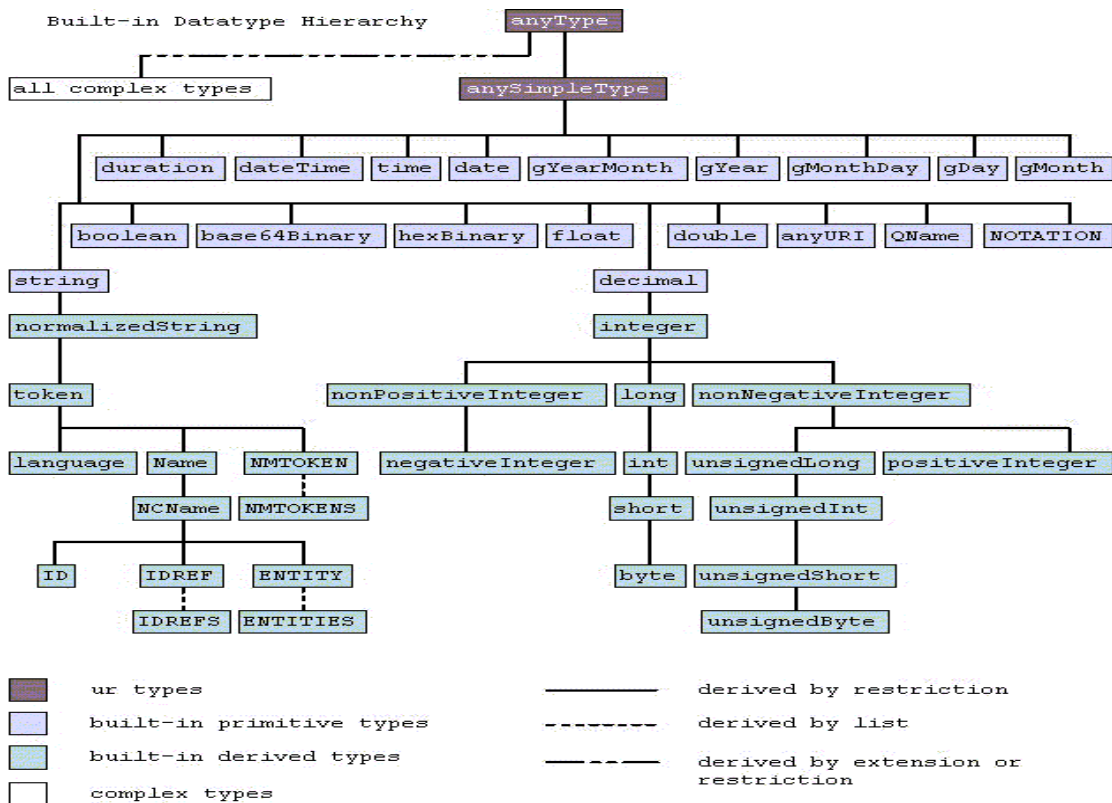
Beispiel:

```
<xsd:simpleType name="Studenten">
    <xsd:list itemType="xsd:string"/>
</xsd:simpleType>
```

Die Vereinigungstypen werden als Unions bezeichnet. Mit Hilfe der Unions kann man unterschiedliche Datentypen vereinigen. Hierbei gibt es zwei verschiedene Methoden. Man kann die Datentypen, welche das Union beinhalten soll, direkt in der Union definieren, oder man benutzt das Attribut "memberTypes", mit dem man auf schon existierende Datentypen verweisen kann. Auf Unions können die Fassetten pattern und enumeration angewendet werden.

Beispiel mit dem memberType Attribut:

```
<xsd:simpleType name="Bsp-Union">
    <xsd:union memberTypes="Typ1 Typ2"/>
</xsd:simpleType>
```



3.2 Komplexe Datentypen

Komplexe Datentypen enthalten normalerweise eine Menge von Elementdeklarationen, Elementreferenzen und Attributdeklarationen. Es gibt komplexe Datentypen mit keinem Inhalt. Diese enthalten keine Kindelemente und werden als "leere Typen" bezeichnet. Außerdem gibt es unter anderem noch komplexe Typen mit gemischtem Inhalt. Dies kann durch das "mixed"-Attribut angegeben werden, und bedeutet, dass im XML-Dokument an dieser Stelle normaler Text und Variablen vorkommen dürfen.

Beispiel:

Dies ist ein Beispiel für einen gemischten Inhalt. Man sieht, wie ein solcher Typ im Schema deklariert wird und wie der dazu passende Teil aus dem Dokument aussehen könnte.

XML-Schema:

```
<xsd:element name="bericht" type="berichtTyp"/>
  <xsd:complexType name="berichtTyp" mixed="true">
    <xsd:sequence>
      <xsd:element name="fett" type="xsd:string" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
```

XML-Dokument:

```
<bericht>
  In diesem <fett>Bericht</fett> sind Teile des Textes
  zwischendrin <fett>auchmal fett</fett> gedruckt.
</bericht>
```

3.2.1 Kompositoren

Um die Kindelemente eines komplexen Typs zu definieren, gibt es drei verschiedene Möglichkeiten:

Choice: Es werden verschiedene Elemente vorgegeben und im XML-Dokument darf nur eines von den aufgelisteten Elementen vorkommen.

Beispiel:

Die Leistung des Motors kann entweder in kW-Leistung angegeben werden oder in PS-Leistung.

```
<xsd:complexType name="motorTyp">
  <xsd:choice>
    <xsd:element name="kWLeistung" type="xsd:positiveInteger"/>
    <xsd:element name="PSLeistung" type="xsd:positiveInteger"/>
  </xsd:choice>
</xsd:complexType>
```

Sequence: Bei der "sequence" werden auch wieder Elemente vorgegeben, diese müssen im XML-Dokument in der angegebenen Reihenfolge vorkommen.

All: Bei dem Kompositor "all" ist die Reihenfolge, in der die Elemente im XML-Dokument auftreten, egal.

Auf die Kompositoren "choice" und "sequence" können die Häufigkeitsbeschränkungen minOccurs und maxOccurs angewendet werden. Bei "all" hingegen nicht, daher können Elemente hier nur null- bis einmal auftreten.

3.2.2 Komplexe Typen ableiten

Es gibt die Möglichkeit, wie auch bei den einfachen Typen, neue komplexe Typen zu definieren indem man entweder einfache Typen oder komplexe Typen ableitet. Dies geschieht entweder durch das Einschränken (restriction) oder durch das Erweitern (extension) eines Typs.

Beim Einschränken wird die Häufigkeit des Auftretens von Elementen verändert. Es ist darauf zu achten, dass der eingeschränkte Typ Teilmenge des Basistyps ist, und dass alle Elemente und Attribute, die im abgeleiteten Typ verwendet werden sollen, erneut deklariert werden müssen.

Beispiel:

In dem folgenden Beispiel haben wir den Basistyp "rezept". Von diesem aus leiten wir den neuen komplexen Typen "schnellrezept" durch Einschränkung ab, indem wir das maximale Auftreten des Elementes "zutat" auf 20 beschränken.

```
<xsd:complexType name="rezept">
  <xsd:element name="zutat" type="zutaten" maxOccurs="unbounded" />
  <xsd:element name="zubereitung" type="anleitung" />
</xsd:complexType>

<xsd:complexType name="schnellrezept">
  <xsd:complexContent>
    <xsd:restriction base="rezept">
      <xsd:element name="zutat" type="zutaten" maxOccurs="20"/>
      <xsd:element name="zubereitung" type="anleitung" />
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

Bei der Erweiterung werden neue Elemente oder Attribute hinzugefügt. Der Basistyp kann hierbei entweder ein einfacher Typ sein oder ein komplexer Typ. Im Gegensatz zur Einschränkung müssen nur die neuen Elemente und Attribute angegeben werden.

Beispiel:

Der Basistyp in diesem Beispiel ist der einfache Typ "decimal". Dieser wird um das Attribut "währung" erweitert.

```
<xsd:element name="PreisInternational">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:decimal">
        <xsd:attribute name="währung" type="xsd:string"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

3.3 Anonyme Typen

Wenn man viele Typen definiert und diese nur einmal referenziert kann es innerhalb eines Schemas schnell unübersichtlich werden. In diesem Fall kann man bei der Deklaration von Elementen und Attributen, statt des Type-Attributs, einen neuen unbekanntem Typ definieren. Dieser Typ wird als anonymer Typ bezeichnet. Die Definition eines solchen anonymen Typs ist immer nur innerhalb einer Element- bzw. Attributdeklaration möglich. Ein anonymer Typ kann nicht referenziert werden.

Beispiel:

Es wird ein Element mit dem Namen Anzahl deklariert. Der Typ wird in diesem Fall nicht durch das Typ-Attribut angegeben, sondern durch einen simplen Typen der direkt in der Elementdeklaration definiert wird.

```
<xsd:element name="Anzahl">
  <xsd:simpleType>
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:maxInclusive value="100"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

3.4 Häufigkeitsbeschränkungen

Es gibt die Möglichkeit, die Häufigkeit des Auftretens von Elementen und Attributen zu beschränken. Man kann dabei unter anderem folgende Attribute verwenden:

minOccurs-Attribut und maxOccurs-Attribut:

Mit dem minOccurs bzw. maxOccurs-Attribut kann man angeben, wie oft ein Element im XML-Dokument mindestens vorkommen muss bzw. wie oft es maximal vorkommen darf. Beide sind standardmäßig auf eins gesetzt. Es muss darauf geachtet werden, dass der Wert des maxOccurs niemals kleiner sein darf als der Wert des minOccurs

use-Attribut:

Des Weiteren gibt es noch das use-Attribut, für welches folgende Werte gesetzt werden können:

required => ein Element/Attribut muss genau einmal vorkommen

optional => ein Element/Attribut kann null- bis einmal vorkommen

prohibited => ein Element/Attribut darf gar nicht vorkommen. Das prohibited Attribut ist z.B. sinnvoll, wenn man einen Typen ableiten möchte und einzelne Elemente oder Attribute nicht mehr benötigt.

fixed-Attribut:

Man kann einen festen Wert für ein Element/Attribut setzen. Das Element/Attribut muss im XML-Dokument genau diesen Wert haben. Wenn das nicht der Fall ist, ist das Dokument nicht konform zum Schema.

default-Attribut:

Mit dem default-Attribut kann man einen Wert für ein Element/Attribut setzen. Bekommt das Element/Attribut im XML-Dokument einen anderen Wert, dann wird der default-Wert überschrieben.

3.5 Attribute

Attribute dürfen immer nur einen einfachen Typ haben. Sie werden immer nach den Elementdefinitionen definiert. Es ist darauf zu achten, dass Attribute innerhalb eines Elementes nicht doppelt vorkommen dürfen. Auf Attribute können die Häufigkeitsbeschränkungen `use`, `fixed` und `default` angewendet werden. Die Häufigkeitsbeschränkungen `min-` und `maxOccurs` machen hingegen weniger Sinn, da ein Attribut immer nur null- bis einmal vorkommen darf. Attribute können in sogenannte Attributgruppen zusammengefasst werden.

3.6 Globale Elemente und Attribute

Elemente und Attribute können auch global deklariert werden. Sie dürfen dann aber weder Referenzen noch Kardinalitäten enthalten. Unter der Verwendung des `"ref"`-Attributs kann auf sie zugegriffen werden. Es ist zu beachten, dass nur global deklarierte Elemente und Attribute referenziert werden dürfen.

4. Wiederverwendbarkeit

Um in einem Schema die Datenstrukturen von anderen Schemata wiederzuverwenden gibt es folgende drei Mechanismen:

`include`: Beim `"include"` werden alle Deklarationen aus dem angegebenen Schema übernommen. Hierbei ist darauf zu achten, dass die Namesräume des Ziel- und Quellschemas gleich sind

```
<include schemaLocation=„URI“/>
```

`redefine`: Das `redefine` funktioniert im Wesentlichen genauso wie das `include`, die Komponenten können aber umdefiniert werden.

```
<redefine schemaLocation=„URI“>
```

`import`: Mit dem `import` können einzelne Elemente importiert werden. Der Namensraum spielt hierbei keine Rolle.

```
<import namespace=„URI“/>
```

5. Quellenangabe

Folgende Quellen habe ich für mein Referat und die Ausarbeitung verwendet:

- XML Schema Eric van der Vlist, Deutsche übersetzung von Gisbert W. Selke, O'Reilly
- <https://www.bg.bib.de/portale/xml/pdf/XML-Schema.pdf>
- <http://www.edition-w3c.de/TR/2001/REC-xmlschema-0-20010502/>
- http://de.wikipedia.org/wiki/XML_Schema