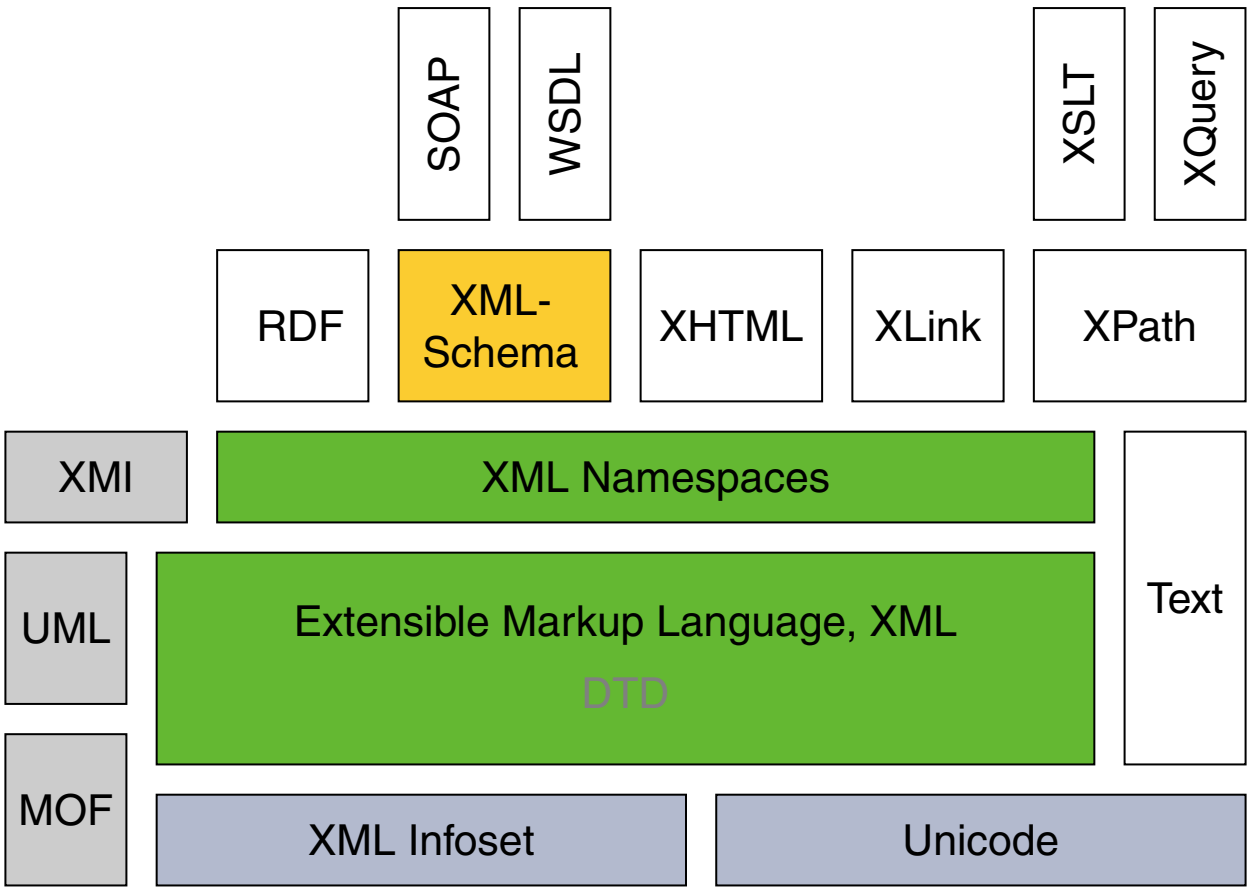
Kapitel WT:III (Fortsetzung)

III. Dokumentsprachen

- Auszeichnungssprachen
- HTML
- Cascading Stylesheets CSS
- XML-Grundlagen
- XML-Schema
- Die XSL-Familie
- Parse-Paradigmen und APIs für XML

XML-Schema

Einordnung [Jeckle 2004]



XML-Schema [W3C reports]

Historie: zentrale XML-Spezifikationen

2006 Extensible Markup Language (XML) 1.1. Recommendation. [W3C REC, status]

2004 XML Schema Part 0: Primer. Recommendation. [W3C REC, status]

2012 XML Schema (XSD) 1.1 Part 1: Structures. [W3C REC]

2012 XML Schema (XSD) 1.1 Part 2: Datatypes. [W3C REC]

2021 XSL Transformations (XSLT) 2.0. Recommendation. [W3C REC, status]

2017 XML Path Language (XPath) 3.1. Recommendation. [W3C REC, status]

2017 XML Query Language (XQuery) 3.1. Recommendation. [W3C REC, status]

2012 XSL Formatting Objects (XSL-FO) 2.0. Working Draft. [W3C WD, Wiki]

Bemerkungen:

- XML-Schema $\equiv$ XML Schema Definition Language, XSD.
- XML-Schema bildet zusammen mit XML 1.1 und den Namensräumen die Basis aller weiteren W3C-XML-Sprachstandards. Die Grammatiken neu entwickelter XML-Sprachen werden nicht mehr in der DTD-Syntax formuliert.
- Der XSD-Sprachvorschlag gliedert sich in zwei Teile:
  1. XSD-Part 1 „Structures“ www.w3.org/TR/xmlschema11-1. Beschreibung von Inhaltsmodellen für Elemente, Attributstrukturen und wiederverwendbaren Strukturen. Bildet die Konzepte von DTDs nach.
  2. XSD-Part 2 „Datatypes“ www.w3.org/TR/xmlschema11-2. Beschreibung von Datentypen für XML-Schemas sowie andere XML-Spezifikationen. Es handelt sich um ein eigenständiges Typsystem, das in mehreren W3C-Arbeitsgruppen Verwendung findet.

  Der XML-Schema Part 0 „Primer“ www.w3.org/TR/xmlschema-0 gibt eine gute Einführung in die Konzepte und Ziele zu XML-Schema.

XML-Schema

DTD versus XML-Schema [Instanz mit DTD]

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

XML-Schema

DTD versus XML-Schema [Instanz mit DTD]

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="vorname" type="xs:string"/>
        <xs:element name="nachname" type="xs:string"/>
        <xs:element name="beruf" type="xs:string"
            minOccurs="0" maxOccurs="3"/>
      </xs:sequence>
      <xs:attribute name="geburtsjahr" type="xs:gYear" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML-Schema

DTD versus XML-Schema [Instanz mit DTD]

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="vorname" type="xs:string"/>
        <xs:element name="nachname" type="xs:string"/>
        <xs:element name="beruf" type="xs:string"
           minOccurs="0" maxOccurs="3"/>
      </xs:sequence>
      <xs:attribute name="geburtsjahr" type="xs:gYear" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML-Schema

DTD versus XML-Schema [Instanz mit DTD]

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="vorname" type="xs:string"/>
        <xs:element name="nachname" type="xs:string"/>
        <xs:element name="beruf" type="xs:string"
           minOccurs="0" maxOccurs="3"/>
      </xs:sequence>
      <xs:attribute name="geburtsjahr" type="xs:gYear" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML-Schema

DTD versus XML-Schema [Instanz mit DTD]

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="vorname" type="xs:string"/>
        <xs:element name="nachname" type="xs:string"/>
        <xs:element name="beruf" type="xs:string"
           minOccurs="0" maxOccurs="3"/>
      </xs:sequence>
      <xs:attribute name="geburtsjahr" type="xs:gYear" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Bemerkungen:

- Ein XML-Schema beinhaltet [W3C]:
  1. *Definitionen* von einfachen und komplexen *Datentypen*.
  2. *Deklarationen* der *Elementtypen*, die in Instanzdokumenten erlaubt sind; ein Elementtyp ist von einem bestimmten Datentyp.

- Zur Deklaration der erlaubten Elementtypen dient das `<xs:element>`-Element des XSD-Vokabulars. Beachte, dass Instanzen des `<xs:element>`-Elements entweder

  (a) eine Typdefinition zwischen öffnendem und schließendem Tag aufnehmen:

```
<xs:element name="person">
  <xs:complexType>
    ...
  </xs:complexType>
</xs:element>
```

  (b) oder leer verwendet werden, weil ein existierender Typ zum Einsatz kommt:

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

  Die Syntax im ersten Fall folgt dem Entwurfsmuster zur Deklaration von Elementtypen einschließlich der Definition eines neuen (hier: anonymen, komplexen) Datentyps.

  Die Syntax im zweiten Fall folgt dem Entwurfsmuster zur Deklaration von Elementtypen unter Rückgriff auf einen bereits definierten Datentyp über das `type`-Attribut.

Bemerkungen: (Fortsetzung)

- Beachte, dass bei XML-Schema die (kontextfreie) Grammatik zur Beschreibung einer Dokumentenstruktur nicht mehr in der Form von Regeln, sondern „objektorientiert“, mittels XSD-Elementinstanzen und deren Schachtelung geschieht.
- XML-Schema ist die Metasprache, um die erlaubten XML-Instanzdokumente zu beschreiben. Gleichzeitig wird XML-Schema als ein XML-(instanz)dokument notiert. Somit ist die Syntax der Metasprache (hier: Grammatik im XML-Schemadokument) gleich der Syntax der Objektsprache (hier: Elemente im XML-Instanzdokument).

XML-Schema

DTD versus XML-Schema (Fortsetzung)

Grenzen von DTDs:

- nur wenige Datentypen, Typsystem nicht erweiterbar
- Syntax nicht XML-konform
- keine Unterstützung von Namensräumen
- keine Möglichkeit zur DTD-Erweiterung
  Eine DTD muss vollständig sein und sämtliche Regeln für ihre Anwendung definieren.
- keine Möglichkeit zur DTD-Modularisierung
  Elementtypen sind nur innerhalb der definierenden DTD wiederverwendbar, Attribute sind an das umgebende Element gebunden.

XML-Schema

DTD versus XML-Schema (Fortsetzung)

Das Typsystem von XML-Schema ermöglicht:

- Definition von Constraints für zugelassenen Inhalt
- Überprüfung der Korrektheit von Daten
- Verarbeitung von Daten aus Datenbanken
- Spezialisierung von Datentypen
- Definition komplexer Datentypen
- Konvertierung zwischen Daten verschiedenen Typs

Weitere Merkmale [w3schools]:

- XML-Schemata verwenden XML-Syntax
- XML-Schemata verwenden das XML-Namensraumkonzept
- XML-Schemata sind erweiterbar, modularisierbar, wiederverwendbar

XML-Schema

DTD versus XML-Schema (Fortsetzung)

Unterscheidung von Dokumenten hinsichtlich ihres Aufbaus:

1. Erzählende (*narrative*) Dokumente.
   Dokumente, die aus Abschnitten und Unterabschnitten bestehen; die gesamte Struktur ist weitgehend *linear*: Bücher, Artikel, etc.

2. Datensatzartige Dokumente.
   Stark typisierte Dokumente; zielen auf Datenaustausch ab.

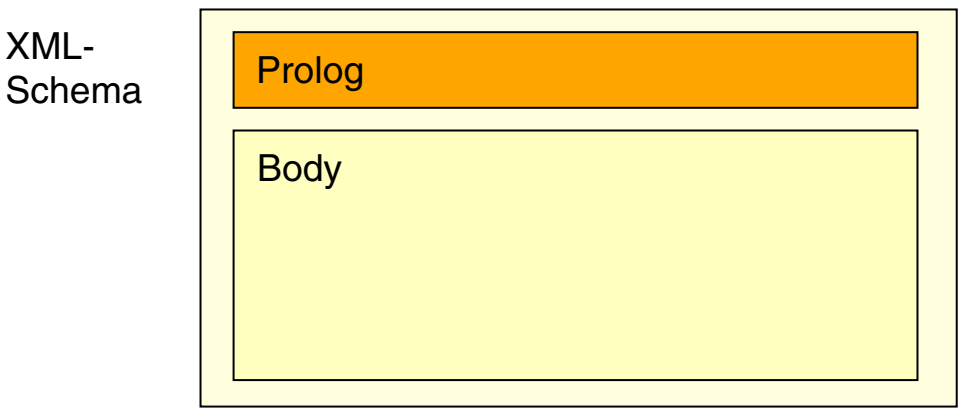
DTDs sind/waren gut für erzählende Dokumente geeignet,

XML-Schema ist optimiert für datensatzartige Dokumente.

XML-Schema

Aufbau XML-Schema

XML-Schemata sind XML-Dokumente:



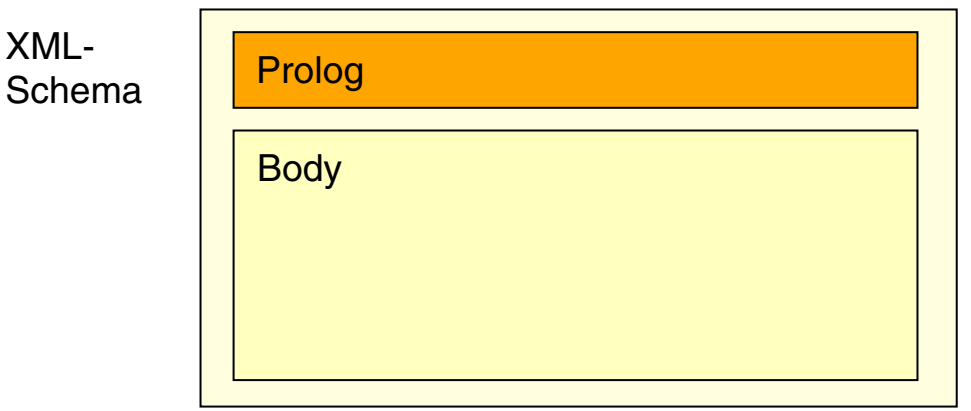

```
<?xml version="1.0"?>

<xs:schema xmlns:xs=...>
  <xs:element name="person">
    ...
  </xs:element>
  ...
  <xs:element name=...>
    ...
  </xs:element>
</xs:schema>
```

- Wurzelelement jedes XML-Schemas ist das Element `<xs:schema>`.
- Die direkten (= nicht tiefer geschachtelt liegenden) Kindelemente von `<xs:schema>` Elemente und Attribute sind global.
- Die Attribute von `<xs:schema>` definieren Eigenschaften, die für alle Elemente und Attribute des Schemas gelten.
- Vergleiche hierzu die XML-Dokumentstruktur.

XML-Schema

Aufbau XML-Schema

XML-Schemata sind XML-Dokumente:




```
<?xml version="1.0"?>

<xs:schema xmlns:xs=...>
  <xs:element name="person">
    ...
  </xs:element>
  ...
  <xs:element name=...>
    ...
  </xs:element>
</xs:schema>
```

- Wurzelelement jedes XML-Schemas ist das Element `<xs:schema>`.
- Die direkten (= nicht tiefer geschachtelt liegenden) Kindelemente von `<xs:schema>` Elemente und Attribute sind global.
- Die Attribute von `<xs:schema>` definieren Eigenschaften, die für alle Elemente und Attribute des Schemas gelten.
- Vergleiche hierzu die XML-Dokumentstruktur.

Bemerkungen:

- Globale Elemente können als Wurzelelement eines Instanzdokuments (= als Dokumentelement) verwendet werden.
- Die Dateiendung einer XML-Schemadatei ist `.xsd`.

XML-Schema

Aufbau XML-Schema (Fortsetzung) [DTD vs Schema] [Instanz]

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

    elementFormDefault="qualified">

    <xs:element name="person">

       <xs:complexType>

              ...

       </xs:complexType>

    </xs:element>

</xs:schema>
```

XML-Schema

Aufbau XML-Schema (Fortsetzung) [DTD vs Schema] [Instanz]

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

    elementFormDefault="qualified">

    <xs:element name="person">

       <xs:complexType>

              ...

       </xs:complexType>

    </xs:element>

</xs:schema>
```

XML-Schema

Aufbau XML-Schema (Fortsetzung) [DTD vs Schema] [Instanz]

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

    elementFormDefault="qualified">

    <xs:element name="person">

       <xs:complexType>

              ...

       </xs:complexType>

    </xs:element>

</xs:schema>
```

XML-Schema

Aufbau XML-Schema (Fortsetzung) [DTD vs Schema] [Instanz]

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

    elementFormDefault="qualified">
    <xs:element name="person">
       <xs:complexType>
              ...
       </xs:complexType>
    </xs:element>
</xs:schema>
```

XML-Schema

Aufbau XML-Schema (Fortsetzung) [DTD vs Schema] [Instanz]

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="https://webtec.webis.de"
   elementFormDefault="qualified">

   <xs:element name="person">

      <xs:complexType>
            ...
      </xs:complexType>

   </xs:element>

</xs:schema>
```

XML-Schema

Aufbau XML-Schema (Fortsetzung) [DTD vs Schema] [Instanz]

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="https://webtec.webis.de"
    elementFormDefault="qualified">

    <xs:element name="person">

        <xs:complexType>
                ...
        </xs:complexType>

    </xs:element>

</xs:schema>
```

Zwei Arten von Namensräumen sind zu deklarieren:

1. Einen oder mehrere Namensräume, zu denen die *im Schema verwendeten* Elemente und Datentypen (= XSD-Vokabular) gehören.
2. Einen Zielnamensraum (*Target Namespace*) für die global deklarierten Elemente und Attribute des Schemas (= Autorenvokabular). Dieser dient zur Unterscheidung von Elementdeklarationen in anderen Vokabularen. [W3C]

XML-Schema

Aufbau XML-Schema (Fortsetzung) [DTD vs Schema] [Instanz]

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="https://webtec.webis.de"
    elementFormDefault="qualified">
    <xs:element name="person">
        <xs:complexType>
                ...
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Deklaration von `person` als Wurzel für Instanzdokumente.

Zwei Arten von Namensräumen sind zu deklarieren:

1. Einen oder mehrere Namensräume, zu denen die *im Schema verwendeten* Elemente und Datentypen (= XSD-Vokabular) gehören.
2. Einen Zielnamensraum (*Target Namespace*) für die global deklarierten Elemente und Attribute des Schemas (= Autorenvokabular). Dieser dient zur Unterscheidung von Elementdeklarationen in anderen Vokabularen. [W3C]

XML-Schema

Aufbau XML-Schema (Fortsetzung) [DTD vs Schema] [Instanz]

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="https://webtec.webis.de"
    elementFormDefault="qualified">

    <xs:element name="person">

        <xs:complexType>
                ...
        </xs:complexType>

    </xs:element>

</xs:schema>
```

Definition eines anonymen, komplexen Datentyps für `person`.

Zwei Arten von Namensräumen sind zu deklarieren:

1. Einen oder mehrere Namensräume, zu denen die *im Schema verwendeten* Elemente und Datentypen (= XSD-Vokabular) gehören.
2. Einen Zielnamensraum (*Target Namespace*) für die global deklarierten Elemente und Attribute des Schemas (= Autorenvokabular). Dieser dient zur Unterscheidung von Elementdeklarationen in anderen Vokabularen. [W3C]

Bemerkungen:

- Syntax und Semantik des XSD-Vokabulars sind durch die normativen Referenzen XML-Schema Part 0, Part 1 und Part 2 des W3C standardisiert.

  Das XSD-Vokabular enthält die Namen für 36 Elemente und 28 Attribute, die zur Erstellung von XML-Schemadokumenten zur Verfügung stehen:

  - `xs:all`
  - `xs:annotation`
  - `xs:any`
  - `xs:anyAttribute`
  - `xs:appinfo`
  - `xs:attribute`
  - `xs:attributeGroup`
  - `xs:choice`
  - `xs:complexContent`
  - `xs:complexType`
  - ⋮
  - `xs:abstract`
  - `xs:attributeFormDefault`
  - `xs:base`
  - `xs:block`
  - ⋮

  Der zugehörige Namensraum heißt http://www.w3.org/2001/XMLSchema. Das übliche Präfix bei der Namensraumdeklaration ist `xs:`, es kann aber beliebig gewählt werden.

- Alle im Schema global notierten Elemente und Attribute sind dem Zielnamensraum zugeordnet. Bei ihrer Instanziierung in einem schemakonformen Instanzdokument müssen sie zu dem entsprechenden Namensraum gehören, also entsprechend qualifiziert sein. Auch bei Referenzen innerhalb des Schemas muss diese Namensraumzugehörigkeit beachtet werden.

- Das `targetNamespace`-Attribut, also die Deklaration eines Zielnamensraums, ist optional.

- Die Einbettung eines XML-Schemas in ein Instanzdokument, also die Bildung von “Internal Subsets” wie bei DTDs, ist nicht möglich.

Bemerkungen: (Fortsetzung)

- Durch Angabe der Attribute `elementFormDefault` und `attributeFormDefault` lässt sich die Zuordnung der *lokal definierten* Elemente und Attribute zum Zielnamensraum steuern.

  Standardmäßig sind diese Attribute auf `unqualified` gesetzt, und dann bezieht sich ein deklarierter Zielnamensraum nur auf die globalen Elemente und Attribute. Wird der Wert der beiden Attribute auf `qualified` gesetzt, so sind auch die lokalen Elemente und Attribute als `qualified` deklariert und müssen im Instanzdokument entsprechend qualifiziert verwendet werden.

- Um die Wartbarkeit und Lesbarkeit großer Schemata zu verbessern, kann ein Schema auf mehrere Schemadateien (mit dem gleichen Zielnamensraum) aufgeteilt werden. Die einzelnen Dateien werden dann mittels des Elements `include` aus dem XSD-Vokabular in einem Hauptdokument zusammengefasst.

- Mit dem Element `import` aus dem XSD-Vokabular lassen sich – unter Angabe eines neuen Zielnamensraums – Elemente aus anderen Schemata importieren.

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument [Schema]

```
<?xml version="1.0"?>

<person
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://webtec.webis.de person.xsd"
    xmlns="https://webtec.webis.de">

  <vorname>Alan</vorname>
  <nachname>Turing</nachname>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument [Schema]

```
<?xml version="1.0"?>

<person
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://webtec.webis.de person.xsd"
    xmlns="https://webtec.webis.de">

  <vorname>Alan</vorname>
  <nachname>Turing</nachname>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument [Schema]

```
<?xml version="1.0"?>

<person
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://webtec.webis.de person.xsd"
    xmlns="https://webtec.webis.de">

  <vorname>Alan</vorname>
  <nachname>Turing</nachname>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument [Schema]

```
<?xml version="1.0"?>

<person
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://webtec.webis.de person.xsd"
    xmlns="https://webtec.webis.de">

  <vorname>Alan</vorname>
  <nachname>Turing</nachname>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

- Die Attribute `schemaLocation` bzw. `noNamespaceSchemaLocation` dienen zur Referenz auf das Schema. Genau eines muss angegeben sein – abhängig davon, ob das Schema einen Zielnamensraum festlegt oder nicht.
- Der erste Parameter von `schemaLocation` ist eine URI für den Zielnamensraum. Der zweite Parameter ist eine relative oder absolute URL, die angibt, wo das Schema liegt.
- `noNamespaceSchemaLocation` hat nur den URL-Parameter für das Schema.

Bemerkungen:

- XSD-Part 1 “Structures” definiert auch Vokabular für XML-Instanzdokumente; es handelt sich um insgesamt vier Attribute:
  - `xsi:type`
  - `xsi:nil`
  - `xsi:schemaLocation`
  - `xsi:noNamespaceSchemaLocation`

  Der zugehörige Namensraum heißt http://www.w3.org/2001/XMLSchema-instance. Das übliche Präfix bei der Namensraumdeklaration ist `xsi:`, es kann aber beliebig gewählt werden.

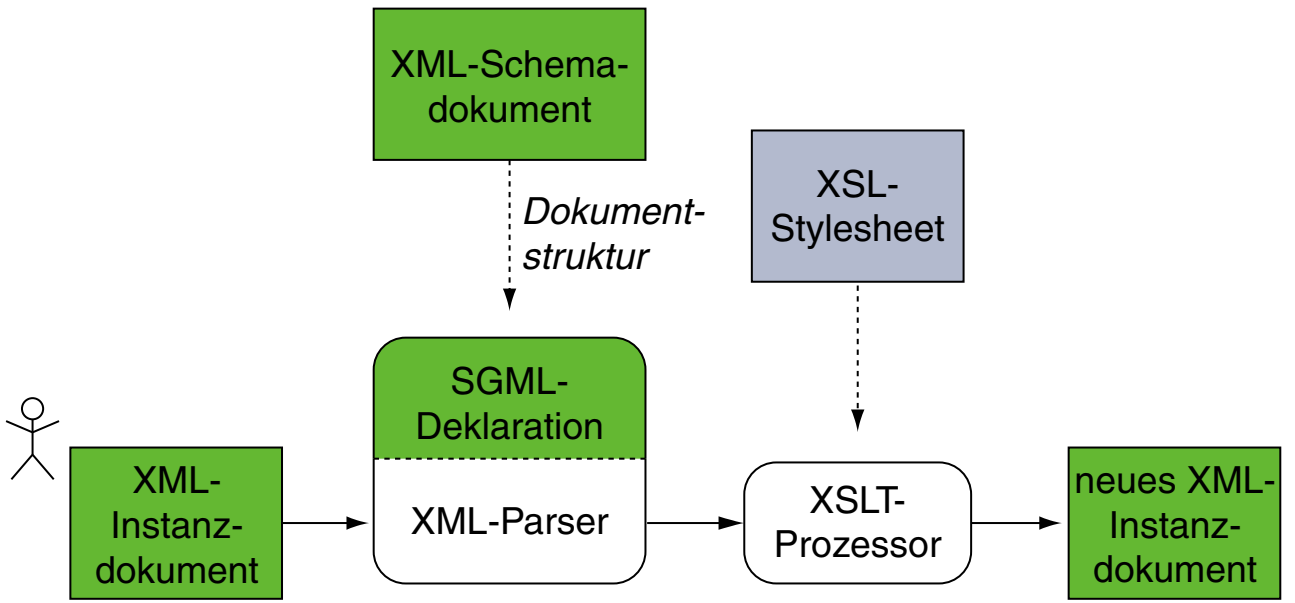
- Im Beispiel wird der Namensraum für Instanzdokumente deklariert und an `xsi:` gebunden, um das Attribut `schemaLocation` zu qualifizieren – das Attribut also vollständig zu benennen und so dem Parser (bspw. im Web-Browser) bekannt zu machen.

- Um schemakonform zu sein, muss der erste Parameter des Attributes `schemaLocation` im XML-Instanzdokument mit dem Wert des Attributes `targetNamespace` im XML-Schema übereinstimmen. Legt das XML-Schema *keinen* Zielnamensraum fest, so wird eine Referenz auf dieses Schema mittels `noNamespaceSchemaLocation` spezifiziert. [W3C]

- Im Beispiel wird mit `xmlns="https://webtec.webis.de"` ein Default-Namensraum eingeführt, der dem Zielnamensraum `https://webtec.webis.de` entspricht. Somit gehört das `<person>`-Element im XML-Instanzdokument auch ohne Präfix zum Zielnamensraum und ist schemakonform instanziiert.

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument (Fortsetzung)



Vergleiche hierzu

- die SGML Dokumentenverarbeitung,
- die HTML Dokumentenverarbeitung,
- und die XML Dokumentenverarbeitung mit DTDs.

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument (Fortsetzung)

Definition 11 (gültig hinsichtlich Schema, Instanzdokument [vgl. DTD])

Ein XML-Dokument heißt gültig hinsichtlich eines Schemas (*schema valid*), wenn es über ein Schema verfügt, und konform zu diesem aufgebaut ist.

Das zu validierende Dokument wird als Instanzdokument bezeichnet.

Bemerkungen:

- Aufgrund der Realisierung der Schemasprache als XML-Sprache ist jedes Schema auch ein XML-Dokument. Dadurch wird es möglich, alle erlaubten Schemata selbst durch ein Schema – ein sogenanntes *Metaschema* – zu beschreiben und zu validieren. [W3C]

  Somit können Schemata mit denselben Werkzeugen analysiert, verarbeitet und geprüft werden, die auch für Instanzdokumente Verwendung finden.

- Für ein Metaschema kann wiederum ein Schema angegeben werden. Um eine „unendliche“ Reihung zur Validierung notwendiger Schemata zu vermeiden, hat man bei der XML-Standardisierung darauf geachtet, dass die Sprache zur Beschreibung von XML-Schemata ausdrucksstark genug ist, um als Metasprache zur Beschreibung aller erlaubten Schemata zu fungieren.

- Der angegebene Gültigkeitsbegriff lässt die Konformität hinsichtlich einer eventuell existierenden DTD außer Acht. So sind Dokumente denkbar, die zwar hinsichtlich einer gegebenen DTD als valide eingestuft werden, jedoch ein zugehöriges Schema verletzen – und umgekehrt.

Bemerkungen: (Fortsetzung)

- Online-Services und Werkzeuge:
  - www.freeformatter.com/xml-validator-xsd.html (Schemavalidierung)
  - www.utilities-online.info/xsdvalidation (Schemavalidierung)
  - www.altova.com/download-trial.html (allgemein)

- Beispiel zum Testen:
  - Instanzdokument: `personen.xml`,
  - Schemadoukment: `personen.xsd`

- Shell-Aufruf mittels validierendem Parser aus dem Apache Xerces Projekt:

  `[user@pc]$ java dom.Writer -v -s` *Instanzdokument*

XML-Schema

Entwurf eines XML-Schemas

Voraussetzungen, um ein neues XML-Schema zu entwerfen:

- Verständnis für den Zusammenhang zwischen
  - Inhaltsmodell: charakterisiert die Struktur von Elementinstanzen
  - Elementtyp: implementiert ein Inhaltsmodell, ist von einem Datentyp
  - Datentyp eines Elementtyps: deklariert erlaubte Daten
- Sicherer Umgang mit (Ziel-, Default-) Namensräumen

XML-Schema

Entwurf eines XML-Schemas

Voraussetzungen, um ein neues XML-Schema zu entwerfen:

- Verständnis für den Zusammenhang zwischen
  - Inhaltsmodell: charakterisiert die Struktur von Elementinstanzen
  - Elementtyp: implementiert ein Inhaltsmodell, ist von einem Datentyp
  - Datentyp eines Elementtyps: deklariert erlaubte Daten
- Sicherer Umgang mit (Ziel-, Default-) Namensräumen

Aufgaben bei dem Entwurf eines XML-Schemas:

1. Definition neuer Datentypen sowie die Deklaration der in Instanzdokumenten erlaubten Elementtypen.
2. Anwendung eines Entwurfsmusters: anonyme versus benannte Datentypen
3. Optimierung durch Anwendung weiterer Konzepte (Vererbung, Ableitung, etc.) bei der Definition neuer Datentypen.

Bemerkungen:

- W3C-Definitionen von Inhaltsmodell:
  - “A *content model* is a simple grammar governing the allowed types of the child elements and the order in which they are allowed to appear.” [W3C XML REC]
  - “A particle can be used in a complex type definition to constrain the validation of the children of an element information item; such a particle is called a *content model*.”

    “A particle is a term in the grammar for element content, consisting of either an element declaration, a wildcard or a model group, together with occurrence constraints.” [W3C XSD Part 1]
  - Siehe auch: “Building Content Models” [W3C XSD Primer]

XML-Schema

Entwurf eines XML-Schemas (Fortsetzung) [Muster 1 vs 2]

Zwei Entwurfsmuster zur Deklaration von Elementtypen:

1. `<xs:element name="`*Elementname*`">` *Typedefinition* `</xs:element>`
   Deklaration des Elementtyps *Elementname* einschließlich der Definition eines Datentyps.

2. `<xs:element name="`*Elementname*`" type="`*Typename*`"/>`
   Deklaration des Elementtyps *Elementname* unter Verwendung eines existierenden Datentyps.

XML-Schema

Entwurf eines XML-Schemas (Fortsetzung) [Muster 1 vs 2]

Zwei Entwurfsmuster zur Deklaration von Elementtypen:

1. `<xs:element name="`*Elementname*`">` *Typedefinition* `</xs:element>`
   Deklaration des Elementtyps *Elementname* einschließlich der Definition eines Datentyps.

2. `<xs:element name="`*Elementname*`" type="`*Typename*`"/>`
   Deklaration des Elementtyps *Elementname* unter Verwendung eines existierenden Datentyps.

Definition von Datentypen:

- `<xs:complexType> ... </xs:complexType>`
  Definition eines anonymen, komplexen Datentyps für Instanzen von *Elementname*.
  (innerhalb eines `<xs:element name="`*Elementname*`">`-Elementes)

XML-Schema

Entwurf eines XML-Schemas (Fortsetzung) [Muster 1 vs 2]

Zwei Entwurfsmuster zur Deklaration von Elementtypen:

1. `<xs:element name="`*Elementname*`">` *Typedefinition* `</xs:element>`
   Deklaration des Elementtyps *Elementname* einschließlich der Definition eines Datentyps.

2. `<xs:element name="`*Elementname*`" type="`*Typename*`"/>`
   Deklaration des Elementtyps *Elementname* unter Verwendung eines existierenden Datentyps.

Definition von Datentypen:

- `<xs:complexType> ... </xs:complexType>`
  Definition eines anonymen, komplexen Datentyps für Instanzen von *Elementname*.
  (innerhalb eines `<xs:element name="`*Elementname*`">`-Elementes)

- `<xs:complexType name="`*Typename*`"> ... </xs:complexType>`
  Definition eines benannten, komplexen Datentyps mit dem Namen *Typename*.
  (global, direkt unterhalb `<xs:schema>`)

- `<xs:simpleType name="`*Typename*`"> ... </xs:simpleType>`
  Definition eines benannten, einfachen Datentyps mit dem Namen *Typename*.
  (global, direkt unterhalb `<xs:schema>`)

Bemerkungen:

- Entwurfsmuster 1 zur Deklaration von Elementtypen kommt zur Anwendung, wenn kein passender Datentyp vorhanden ist und dieser *inline* (*ad-hoc*) und folglich *anonym* definiert wird.

  Entwurfsmuster 2 zur Deklaration von Elementtypen kommt zur Anwendung, wenn ein passender Datentyp vordefiniert (*built-in*) ist oder bereits an anderer Stelle von einem Autor definiert wurde.

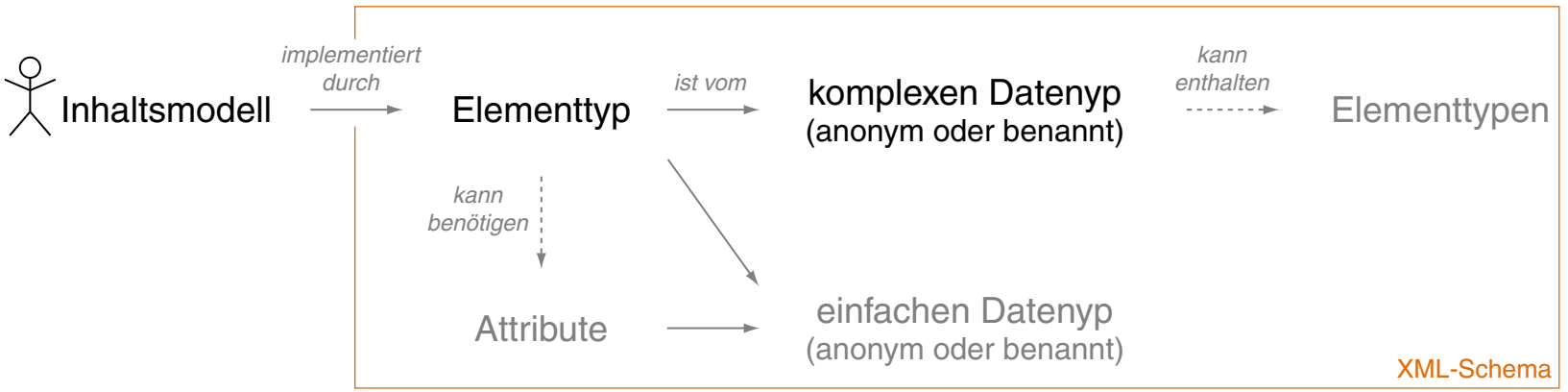
- Die Definition anonymer Datentypen ist sinnvoll, wenn ein Datentyp nur einmal benutzt und der Overhead der Benennung gespart werden soll. [W3C]

  Die Definition benannter Datentypen ist sinnvoll, wenn diese mehrfach (mittels des `type`-Attributs) Verwendung finden sollen.

- Zwischen der Definition von Datentypen und der Deklaration von Elementtypen ist sorgfältig zu unterscheiden. Das Erstgenannte legt Aufbau und Wertebereich eines neuen Datentyps fest; das Zweitgenannte macht die erlaubten Elementinstanzen in schemakonformen Instanzdokumenten bekannt. [W3C]

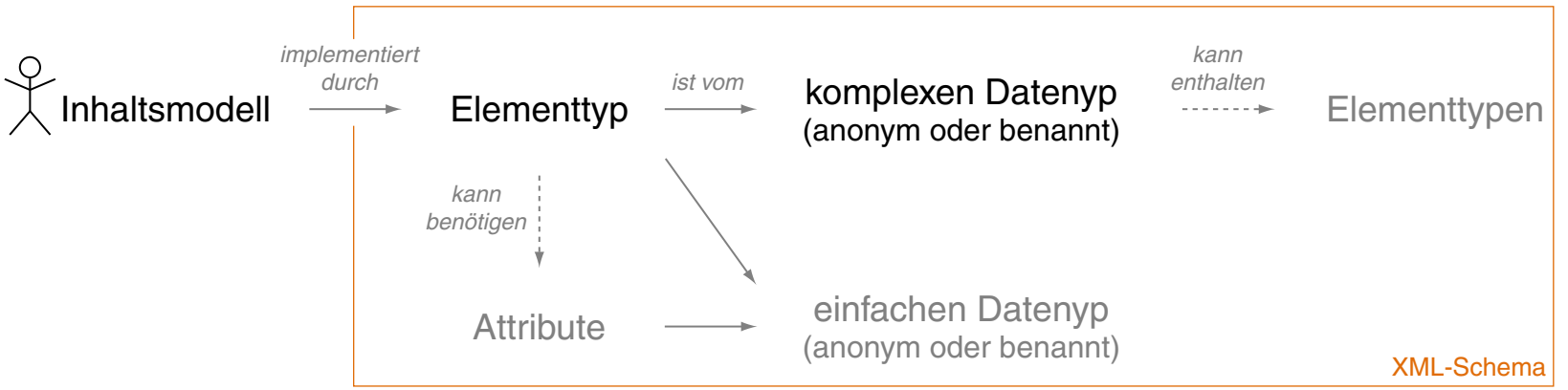
XML-Schema

XML-Schema Teil 1: Inhaltsmodelle [W3C]



XML-Schema

XML-Schema Teil 1: Inhaltsmodelle [W3C]



Wichtige Inhaltsmodelle für Elementtypen [WT:III DTD]:

Inhaltsmodell	Typischer Aufbau
(a) einfacher Inhalt	unstrukturierter, typisierter Text
(b) explizite Kindelemente	vorgeschriebene Schachtelungsstruktur
(c) gemischter Inhalt	Elemente mit unstrukturiertem Text
(d) beliebiger Inhalt	keine Constraints
(e) leerer Inhalt	

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(a) einfacher Inhalt [W3C]:

```
<xs:element
  name="vorname"
  type="xs:string"
  minOccurs="1"
  maxOccurs="3"/>

<xs:element
  name="Elementname"
  type="Typename"/>
```

Vergleichbare DTD-Syntax, ohne Typ-Deklaration:

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

Beispiel einer Elementinstanz:

```
<vorname>Tim</vorname>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(a) einfacher Inhalt [W3C]:

```
<xs:element
  name="vorname"
  type="xs:string"
  minOccurs="1"
  maxOccurs="3"/>

<xs:element
  name="Elementname"
  type="Typename"/>
```

Vergleichbare DTD-Syntax, ohne Typ-Deklaration:

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

Beispiel einer Elementinstanz:

```
<vorname>Tim</vorname>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

Wichtige Attribute in der Deklaration eines Elementtyps:

Attribut	Semantik
default	Zeichenkette mit Default-Wert, der konform zum gewählten Datentyp ist
minOccurs	Mindestanzahl zulässiger Instanzen dieses Elementtyps, Default ist 1
maxOccurs	Höchstanzahl zulässiger Instanzen dieses Elementtyps, Default ist 1
name	unqualifizierter Name des Elementtyps
ref	Verweis auf eine globale Deklaration des Elementtyps
type	Deklaration des Datentyps für den Elementtyp

Bemerkungen:

- In DTDs ist für einfache Inhaltsmodelle mit unstrukturiertem Text nur der Datentyp `#PCDATA` vorgesehen. XML-Schema Part 2 definiert 44 primitive Datentypen. [W3C]
- Mit der Spezifikation `maxOccurs="unbounded"` wird eine beliebige Anzahl an Instanzen zugelassen.

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(b) explizite Kindelemente [W3C] [DTD vs Schema]:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="vorname" type="xs:string" maxOccurs="3"/>
      <xs:element name="nachname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Vergleichbare DTD-Syntax, ohne Anzahl-Constraint:

```
<!ELEMENT person (vorname+, nachname)>
```

Beispiel einer Elementinstanz:

```
<person>
  <vorname>Tim</vorname>
  <nachname>Berners-Lee</nachname>
</person>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(b) explizite Kindelemente [W3C] [DTD vs Schema]:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="vorname" type="xs:string" maxOccurs="3"/>
      <xs:element name="nachname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Vergleichbare DTD-Syntax, ohne Anzahl-Constraint:

```
<!ELEMENT person (vorname+, nachname)>
```

Beispiel einer Elementinstanz:

```
<person>
  <vorname>Tim</vorname>
  <nachname>Berners-Lee</nachname>
</person>
```

Bemerkungen:

- Das Beispiel folgt Entwurfsmuster 1: Deklaration eines Elementtyps einschließlich der Definition eines neuen (anonymen, komplexen) Datentyps.
- Das Inhaltsmodell mit explizit angegebenen Kindelementen stellt das wichtigste Inhaltsmodell für die Modellierungspraxis dar.
- Alternativ zu `<xs:sequence>` zur Angabe der Reihenfolge von Kindelementen kann die Angabe `<xs:choice>` für eine exklusive Auswahl oder `<xs:all>` für eine Liste von Kindelementen in beliebiger Reihenfolge (jedes höchstens einmal) erfolgen.
- Die angegebene Deklaration des Elementtyps ist eine Abkürzung der folgenden Deklaration, bei der ein komplexes Inhaltsmodell (`<xs:complexContent>`) mittels `<xs:restriction>` eines allgemeineren Typs `<xs:anyType>` notiert ist:

```
<xs:element name="person">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:element name="vorname" type="xs:string" maxOccurs="3"/>
          <xs:element name="nachname" type="xs:string"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(c) gemischter Inhalt [W3C]:

```
<xs:element name="Brief">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="Anrede" type="xs:string"/>
      <xs:element name="PS" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Vergleichbare DTD-Syntax:

```
<!ELEMENT Brief (#PCDATA | Anrede | PS)*>
```

Beispiel einer Elementinstanz:

```
<Brief>
<Anrede>Hi Tim</Anrede>
Die W3C-Empfehlung für XSD bringt nicht nur Verbesserungen gegenüber
DTDs. <PS>Viele Grüße</PS>
</Brief>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(c) gemischter Inhalt [W3C]:

```
<xs:element name="Brief">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="Anrede" type="xs:string"/>
      <xs:element name="PS" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Vergleichbare DTD-Syntax:

```
<!ELEMENT Brief (#PCDATA | Anrede | PS)*>
```

Beispiel einer Elementinstanz:

```
<Brief>
<Anrede>Hi Tim</Anrede>
Die W3C-Empfehlung für XSD bringt nicht nur Verbesserungen gegenüber
DTDs. <PS>Viele Grüße</PS>
</Brief>
```

Bemerkungen:

- Das Beispiel folgt Entwurfsmuster 1: Deklaration eines Elementtyps einschließlich der Definition eines neuen (anonymen, komplexen) Datentyps.
- Ein gemischtes Inhaltsmodell (*Mixed Content Model*) liegt vor, wenn die Instanz eines Elementtyps sowohl einfachen Inhalt (unstrukturierten Text) als auch Kindelemente aufnehmen kann. Vergleiche die W3C-Definition von Mixed Content:

  “An element type has mixed content when elements of that type may contain character data, optionally interspersed with child elements.” [W3C]
- Während gemischte Inhalte aus Auszeichnungssymbolen und freiem Text durch DTD-validierende Parser nur rudimentär geprüft werden können, ermöglicht XSD eine vollständige Validierung gemischter Inhalte. Die Strukturvalidierung der DTD erlaubt zwar die Spezifikation von innerhalb unstrukturierter Textpassagen auftretenden Elementen, nicht jedoch die Überwachung deren Auftrittshäufigkeit oder Reihenfolge. [Jeckle 2004]

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(d) beliebiger Inhalt [W3C]:

```
<xs:element
  name="Elementname"
  type="xs:anyType">
</xs:element>
```

bzw.

```
<xs:element
  name="Elementname">
</xs:element>
```

oder

```
<xs:element name="Elementname"/>
```

Vergleichbare DTD-Syntax:

```
<!ELEMENT Elementname ANY>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(d) beliebiger Inhalt [W3C]:

```
<xs:element
  name="Elementname"
  type="xs:anyType">
</xs:element>
```

bzw.

```
<xs:element
  name="Elementname">
</xs:element>
```

oder

```
<xs:element name="Elementname"/>
```

Vergleichbare DTD-Syntax:

```
<!ELEMENT Elementname ANY>
```

Bemerkungen:

- Wird das `type`-Attribut nicht spezifiziert oder – alternativ – explizit der Wert `xs:anyType` zugewiesen, so können Instanzen dieses Elementtyps beliebige, XML-wohlgeformte Inhalte aufnehmen.

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(e) leerer Inhalt [W3C]:

```
<xs:element name="internationalPrice">
  <xs:complexType>
    <xs:attribute name="currency" type="xs:string"/>
    <xs:attribute name="value" type="xs:decimal"/>
  </xs:complexType>
</xs:element>

<xs:element
  name="Elementname">
  <xs:complexType/>
</xs:element>
```

Vergleichbare DTD-Syntax:

```
<!ELEMENT Elementname EMPTY>
```

Beispiel einer Elementinstanz:

```
<internationalPrice currency="EUR" value="423.46"/>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(e) leerer Inhalt [W3C]:

```
<xs:element name="internationalPrice">
  <xs:complexType>
    <xs:attribute name="currency" type="xs:string"/>
    <xs:attribute name="value" type="xs:decimal"/>
  </xs:complexType>
</xs:element>

<xs:element
  name="Elementname">
  <xs:complexType/>
</xs:element>
```

Vergleichbare DTD-Syntax:

```
<!ELEMENT Elementname EMPTY>
```

Beispiel einer Elementinstanz:

```
<internationalPrice currency="EUR" value="423.46"/>
```

Bemerkungen:

- Leere Inhaltsmodelle vermitteln ihre Information ausschließlich über Attribute oder über ihre Position innerhalb anderer Elemente.
- Ein XML-Schema-validierender Parser verhält sich hierbei identisch zu einem DTD-validierenden Parser für das Inhaltsmodell `EMPTY`. D.h., es werden nur die beiden folgenden Darstellungen zur Angabe eines leeren Elements akzeptiert:

  `<elementName/>` `<elementName></elementName>`
- Die angegebene Deklaration des Elementtyps ist eine Abkürzung der folgenden Deklaration, bei der ein komplexes Inhaltsmodell (`<xs:complexContent>`) mittels `<xs:restriction>` eines allgemeineren Typs `<xs:anyType>` notiert ist:

```
<xs:element name="internationalPrice">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:attribute name="currency" type="xs:string"/>
        <xs:attribute name="value" type="xs:decimal"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

XML-Schema

XML-Schema Teil 1: Attribute

Beispiel 1:

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

Innerhalb des `<xs:attribute>`-Elements lassen sich (durch Attribute) spezifische Eigenschaften festlegen, unter anderem:

Attribut	**Semantik**
default	Zeichenkette mit Default-Wert, der konform zum gewählten Datentyp ist
fixed	unveränderliche Wertbelegung
name	unqualifizierter Name des Attributes
ref	Verweis auf eine globale Attributdefinition
type	Deklaration des Datentyps für das Attribut

Vergleiche hierzu die DTD-Attribut-Deklaration.

XML-Schema

XML-Schema Teil 1: Attribute

Beispiel 2:

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="isbn" type="isbnType" use="required"/>
  </xs:complexType>
</xs:element>
```

XML-Schema

XML-Schema Teil 1: Attribute

Beispiel 2:

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="isbn" type="isbnType" use="required"/>
  </xs:complexType>
</xs:element>
```

Beispiel 3:

```
<xs:element name="description">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="source" type="xs:NMTOKEN"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Bemerkungen:

- Attribute dürfen nur von einfachem Typ sein.
- Attribute werden immer am Ende einer Deklaration angegeben, also vor `</xs:extension>`, `</xs:restriction>` oder `</xs:complexType>`.
- Beispiel 3 illustriert, dass auch bei der Deklaration eines Elementtyps ohne Kindelemente eine Attributdeklaration immer dazu führt, dass mittels `</xs:complexType>` ein komplexer Datentyp definiert werden muss.
- In der folgenden, nicht abgekürzten Deklaration für das Beispiel 1 zeigt sich die kanonische Herangehensweise bei der Deklaration von Elementtypen:

```
<xs:element name="book">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:element name="title" type="xs:string"/>
          <xs:element name="author" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="isbn" type="isbnType" use="required"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

XML-Schema

XML-Schema Teil 2: Datentypen [W3C]

Einteilung der Datentypen:

- Primitive Datentypen lassen sich nicht auf Basis anderer Datentypen ableiten. Gegensatz: abgeleitete Datentypen
- Vordefinierte Datentypen (*Built-in Datatypes*) sind Teil der Schema Definition Language XSD [W3C]. Gegensatz: anwenderspezifische Datentypen

XML-Schema

XML-Schema Teil 2: Datentypen [W3C]

Einteilung der Datentypen:

- Primitive Datentypen lassen sich nicht auf Basis anderer Datentypen ableiten. Gegensatz: abgeleitete Datentypen
- Vordefinierte Datentypen (*Built-in Datatypes*) sind Teil der Schema Definition Language XSD [W3C]. Gegensatz: anwenderspezifische Datentypen

Schemaelemente zur Definition anwenderspezifischer Datentypen:

1. `<xs:simpleType>` [W3C]
   Definiert einen neuen einfachen Datentyp für Elemente und Attribute, der ausschließlich aus Text besteht und weder Attribute noch Kindelemente deklarieren darf.
2. `<xs:complexType>` [W3C]
   Definiert einen neuen komplexen Datentyp für Elemente. Kann weitere Elemente (= Kindelemente) deklarieren und Attribute haben.

XML-Schema

XML-Schema Teil 2: Datentypen (Fortsetzung)

Wichtige Schemaelemente, um Constraints für Kindelemente in komplexen Datentypen zu spezifizieren:

Element	Semantik
<xs:all>	jedes Kindelement kann höchstens ein Mal auftreten
<xs:choice>	erlaubt das Auftreten eines der Kindelemente gemäß Attributen
<xs:sequence>	Kindelemente müssen in angegebener Reihenfolge auftreten

Weitere Konzepte zur Definition neuer Datentypen:

- Vererbung
- Ableiten durch Einschränkung
- Ableiten durch Erweiterung
- Formulierung von Wertebereichs-Constraints z.B. durch reguläre Ausdrücke

Bemerkungen:

- Ein Beispiel für einen primitiven Datentyp ist `xs:float`. Dagegen ist `xs:integer` kein primitiver Datentyp, weil er durch Spezialisierung von `xs:decimal` abgeleitet ist. [W3C]
- Alle vier Datentypkombinationen sind möglich: primitive wie auch abgeleitete Datentypen können vordefiniert (*built-in*) oder anwenderspezifisch sein.
- Die Definition anwenderspezifischer Datentypen kann anonym oder benannt geschehen. Die Benennung erfolgt durch das `name`-Attribut im `<xs:complexType>` bzw. `<xs:simpleType>`-Element.

  Anonyme Datentypen beziehen sich nur auf die Deklaration der Elementtypen, in der sie eingeführt wurden. Benannte Datentypen sind für alle tieferliegenden Ebenen sichtbar und lassen sich mittels des `type`-Attributes zur Datentypdeklaration verwenden.
- Bei der Definition anwenderspezifischer Datentypen können innerhalb einer `<xs:complexType>`-Definition noch die Schemaelemente `<xs:simpleContent>` und `<xs:complexContent>` zum Einsatz kommen.

  Sie dienen zur Deklaration des Inhalts im Zusammenhang mit
  - `restriction` (um den Inhalt auf bestimmte Datentypen einzuschränken) bzw.
  - `extension` (um den Inhalt hinsichtlich bestimmter Datentypen zu erweitern).

XML-Schema

Schemaentwurf am Beispiel [Vlist 2001]

Instanzdokument:

Being a Dog is a Full-Time Job

by Charles M. Schulz

Character 1:

Name:	Snoopy
Friend of:	Peppermint Patty
Since:	1990-10-04
Qualification:	extroverted beagle

Character 2:

Name:	Peppermint Patty
Since:	1996-08-22
Qualification:	bold, brash, and tomboyish

XML-Schema

Schemaentwurf am Beispiel [Vlist 2001]

Instanzdokument:

Being a Dog is a Full-Time Job

by Charles M. Schulz

Character 1:

Name:	Snoopy
Friend of:	Peppermint Patty
Since:	1990-10-04
Qualification:	extroverted beagle

Character 2:

Name:	Peppermint Patty
Since:	1996-08-22
Qualification:	bold, brash, and tomboyish

Mögliche DTD:

```
<!ELEMENT book (title, author, character+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT character (name, since?, qualification*)>
...
```

XML-Schema

Schemaentwurf am Beispiel (Fortsetzung)

XML-Source des Instanzdokuments:

```
<?xml version="1.0"?>
<book isbn="0836217462">

  <title>
  <author>

  <character>
    <name>
    <friend-of>
    <since>
    <qualification>
  </character>

  <character>
    <name>
    <since>
    <qualification>
  </character>

</book>
```

XML-Schema

Schemaentwurf am Beispiel (Fortsetzung)

XML-Source des Instanzdokuments:

```
<?xml version="1.0"?>
<book isbn="0836217462">

  <title>Being a Dog Is a Full-Time Job</title>
  <author>Charles M. Schulz</author>

  <character>
    <name>Snoopy</name>
    <friend-of>Peppermint Patty</friend-of>
    <since>1950-10-04</since>
    <qualification>extroverted beagle</qualification>
  </character>

  <character>
    <name>Peppermint Patty</name>
    <since>1966-08-22</since>
    <qualification>bold, brash and tomboyish</qualification>
  </character>

</book>
```

XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“

Deklarationen von Elementtypen isoliert betrachtet:

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“

Deklarationen von Elementtypen isoliert betrachtet:

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:attribute name="isbn" type="xs:string"/>

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

<xs:element name="character" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      ...
      <xs:element name="qualification" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“ (Fortsetzung)

Ineinander geschachtelte Deklarationen von Elementtypen:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>









      </xs:sequence>

    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“ (Fortsetzung)

Ineinander geschachtelte Deklarationen von Elementtypen:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“ (Fortsetzung)

Ineinander geschachtelte Deklarationen von Elementtypen:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
        <xs:element name="character" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              ...
              <xs:element name="qualification" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Bemerkungen:

- Bei dem Entwurfsmuster „Russian Doll Design“ ist das entstehende XML-Schema eng an das Instanzdokument angelehnt; die Elementtypen werden entsprechend ihrer Verwendung im Instanzdokument deklariert.
- Nachteil: Es können tiefgeschachtelte Schemata entstehen, die schwer zu verstehen und zu pflegen sind. Solche Schemata unterscheiden sich in ihrem Aufbau deutlich von den entsprechenden DTDs.

XML-Schema

Entwurfsmuster 1b: Verweise auf globale Deklarationen

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- definition of simple type elements -->
  <xs:element name="title" type="xs:string"/>
  <xs:element name="author" type="xs:string"/>
  <xs:element name="name" type="xs:string"/>
  ...
  <xs:element name="qualification" type="xs:string"/>
  <!-- definition of attributes -->
  <xs:attribute name="isbn" type="xs:string"/>
```

XML-Schema

Entwurfsmuster 1b: Verweise auf globale Deklarationen

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- definition of simple type elements -->
  <xs:element name="title" type="xs:string"/>
  <xs:element name="author" type="xs:string"/>
  <xs:element name="name" type="xs:string"/>
  ...
  <xs:element name="qualification" type="xs:string"/>
  <!-- definition of attributes -->
  <xs:attribute name="isbn" type="xs:string"/>

  <!-- definition of complex type elements -->
  <xs:element name="character">
    <xs:complexType>
      <xs:sequence>
        <!-- simple type elements are referenced with "ref" attribute -->
        <xs:element ref="name"/>
        ...
        <xs:element ref="qualification"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

XML-Schema

Entwurfsmuster 1b: Verweise auf globale Deklarationen (Fortsetzung)

```
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="title"/>
        <xs:element ref="author"/>
        <!-- definition of cardinality when elements are referenced -->
        <xs:element ref="character" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute ref="isbn"/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Bemerkungen:

- Wie bei dem „Russian Doll Design“ werden auch hier die Elementtypen entsprechend ihrer Verwendung im Instanzdokument deklariert. Eine Auffaltung der Schachtelung wird dadurch vermieden, dass – ausgehend von den Blättern (also bottom-up) – für jede Ebene des Dokumentbaums eine Liste der in dieser Ebene neu hinzukommenden Elementtypen erstellt wird.

  Die Deklaration von Elementtypen mit Kindelementen geschieht mittels Verweisen auf Elementtypen aus dieser Liste. Vergleiche hierzu das „Russian Doll Design“, bei dem statt der Verweise die gesamte Deklaration eingebettet wird.

- Man kann die Entwurfsmuster (1a) und (1b) als „Entwurf auf Basis anonymer Datentypen“ bezeichnen: die Deklarationen der Elementtypen für `character` und `book` basieren auf einem direkt (*inline, ad-hoc*) definierten, anonymen Datentyp. Sind mehrere Elementtypen gleich aufgebaut, so besitzen sie bis auf ihren Namen die gleiche Deklaration, was zu Redundanz führt. Die Alternative besteht in der Definition eines benannten Datentyps.

- Durch das Referenzierungskonzept existiert eine erste Möglichkeit zur Wiederverwendung bereits im Schema definierter Elementtypen. Bei dieser einfachen Form der *Textersetzung* werden die Elementtypen literal, also mit ihrem Namen eingebunden. Im Grunde genommen ist diese Art der Wiederverwendung bereits mit den Mitteln von DTDs möglich. [Jeckle 2004]

- “Another authoring style, applicable when all element names are unique within a namespace, is to create schemas in which all elements are global. This is similar in effect to the use of `<!ELEMENT>` in a DTD.” [W3C]

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- definition of named simple types -->
  <xs:simpleType name="nameType">
    <xs:restriction base="xs:string">
      <xs:maxLength value="32"/>
    </xs:restriction>
  </xs:simpleType>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- definition of named simple types -->
  <xs:simpleType name="nameType">
    <xs:restriction base="xs:string">
      <xs:maxLength value="32"/>
    </xs:restriction>
  </xs:simpleType>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- definition of named simple types -->
  <xs:simpleType name="nameType">
    <xs:restriction base="xs:string">
      <xs:maxLength value="32"/>
    </xs:restriction>
  </xs:simpleType>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- definition of named simple types -->
  <xs:simpleType name="nameType">
    <xs:restriction base="xs:string">
      <xs:maxLength value="32"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="sinceType">
    <xs:restriction base="xs:date"/>
  </xs:simpleType>

  <xs:simpleType name="qualificationType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>

  <xs:simpleType name="isbnType">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]10"/>
    </xs:restriction>
  </xs:simpleType>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen (Fortsetzung)

```
<!-- definition of named complex types -->
<xs:complexType name="characterType">
  <xs:sequence>
    <xs:element name="name" type="nameType"/>
    ...
    <xs:element name="qualification" type="qualificationType"/>
  </xs:sequence>
</xs:complexType>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen (Fortsetzung)

```
<!-- definition of named complex types -->
<xs:complexType name="characterType">
  <xs:sequence>
    <xs:element name="name" type="nameType"/>
    ...
    <xs:element name="qualification" type="qualificationType"/>
  </xs:sequence>
</xs:complexType>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen (Fortsetzung)

```
<!-- definition of named complex types -->
<xs:complexType name="characterType">
  <xs:sequence>
    <xs:element name="name" type="nameType"/>
    ...
    <xs:element name="qualification" type="qualificationType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="bookType">
  <xs:sequence>
    <xs:element name="title" type="nameType"/>
    <xs:element name="author" type="nameType"/>
    <xs:element name="character" type="characterType" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="isbn" type="isbnType" use="required"/>
</xs:complexType>




</xs:schema>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen (Fortsetzung)

```
  <!-- definition of named complex types -->
  <xs:complexType name="characterType">
    <xs:sequence>
      <xs:element name="name" type="nameType"/>
      ...
      <xs:element name="qualification" type="qualificationType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="bookType">
    <xs:sequence>
      <xs:element name="title" type="nameType"/>
      <xs:element name="author" type="nameType"/>
      <xs:element name="character" type="characterType" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="isbn" type="isbnType" use="required"/>
  </xs:complexType>

  <!-- The "book" element is of the data type "bookType" -->
  <xs:element name="book" type="bookType"/>

</xs:schema>
```

XML-Schema

Gegenüberstellung der Entwurfsmuster [Muster 1 vs 2]

1. Deklaration eines Elementtyps mit anonymem Datentyp:

```
<xs:element name="bondCar">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="color" type="xs:string"/>
      <xs:element name="enginepower" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

2. Deklaration eines Elementtyps mit benanntem Datentyp:

```
<xs:complexType name="automobile">
  <xs:sequence>
    <xs:element name="color" type="xs:string"/>
    <xs:element name="enginepower" type="xs:integer"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="bondCar" type="automobile"/>
```

XML-Schema

Quellen zum Nachlernen und Nachschlagen im Web: Referenz

- Microsoft. *XML Schemas (XSD) Reference.*
  docs.microsoft.com
- W3C. *XML Schema Part 0: Primer.*
  www.w3.org/TR/xmlschema-0
- W3C. *XML Schema Definition Language (XSD) 1.1 Part 1: Structures.*
  www.w3.org/TR/xmlschema11-1/
- W3C. *XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes.*
  www.w3.org/TR/xmlschema11-2/

XML-Schema

Quellen zum Nachlernen und Nachschlagen im Web: Usage

- Liquid Technologies. *XML Schema Tutorial – 1/5: Defining Elements and Attributes.* www.liquid-technologies.com/xml-schema-tutorial
- Liquid Technologies. *XML Schema Tutorial – 2/5: Best Practices* www.liquid-technologies.com/xml-schema-tutorial
- Vlist. *Using W3C XML Schema.* www.xml.com/pub/a/2000/11/29/schemas/part1.html
- W3 Schools. *XML Schema.* www.w3schools.com/xml/schema_intro.asp