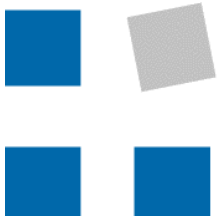

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

XML

Teil 5: XML-Schemas

Abteilung Informatik – WS 02/03

Schema – XML-Schema

- Schema = Dokument, das den Inhalt einer gültigen XML-Dokument beschreibt
- Beispiele von Schemas: DTD, XML-Schema, DDML, Relax
- XML-Schema = Schema, welche in der von der W3C spezifizierten *W3C XML Schema Language* geschrieben wurde
  (genaue Bezeichnung: *W3C XML- Schema*)
- Instanz-Dokument: Dokument, welches durch ein XML-Schema beschrieben wird

XML-Schemas und DTD

- Vorteile der XML-Schemas:
  - Unterstützung von Datentypen
  - Unterstützung von Namensräumen
  - Typen-Ableitung und –Vererbung
- XML-Schemas ersetzen nicht DTDs, sondern ergänzen sie:
  - Entitäten können nicht mit XML-Schemas deklariert werden
  - DTD sind hervorragend für klassische, narrative Dokumente
  - Ein Dokument kann gleichzeitig eine DTD und ein XML-Schema haben

Beispiel: Ein einfaches Schema

XML-Schema (File: `schema.xsd`):

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

Entsprechendes XML-Dokument:

```
<?xml version="1.0"?>
<Text xmlns:xsi=
 "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schema.xsd">
  Hello World!
</Text>
```

XML-Schema-File

- Ein XML-Schema ist ein XML-Dokument
- Erweiterung eines XML-Schema-Files: **`.xsd`** (üblich, nicht obligatorisch)

Struktur eines XML-Schemas

- Root-Element `xs:schema`
- Die Aufzeichnungselemente eines XML-Schemas gehören zum Namensraum `http://www.w3.org/2001/XMLSchema`

  Dieser Namensraum ist meistens mit dem Präfix `xs` oder `xsd` verbunden
- **Globale** Elemente: Elemente, welche durch „**Top-Level**"-Elemente (Child-Elemente des Root-Elements) des Schemas deklariert sind
- Jedes globale Element des Schemas kann das Root eines Instanz-Dokuments werden

XML-Schema → XML-Dokument (1/2)

- Es gibt verschiedene Methoden, um ein XML-Schema einem XML-Dokument zuzuordnen
- Zuordnung eines Schemas, falls sich die Elemente in keinen Namensraum befinden: Attribut **`xsi:noNamespaceSchemaLocation`** im ersten Element hinzufügen, auf das das Schema angewendet wird (meistens: Root-Element)
- Das Präfix **`xsi`** wird auf die URI **`http://www.w3.org/2001/XMLSchema-instance`** abgebildet

XML-Schema: Beispiel 1 (1/2)

```
?xml version="1.0" encoding="UTF-8"?>
SONG xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
         xsi:noNamespaceSchemaLocation="C:\xml\ex51.xsd">
 <TITLE>Yes I Am</TITLE>
 <COMPOSER>Melissa Etheridge</COMPOSER>
 <PRODUCER>Hugh Padgham</PRODUCER>
 <PUBLISHER>Island Records</PUBLISHER>
 <LENGTH>4:24</LENGTH>
 <YEAR>1993</YEAR>
 <ARTIST>Melissa Etheridge</ARTIST>
 <PRICE>$1.25</PRICE>
/SONG>
```

XML-Schema: Beispiel 1 (2/2)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  <xsd:element name="SONG" type="SongType"/>
  <xsd:complexType name="SongType">
    <xsd:sequence>
      <xsd:element name="TITLE" type="xsd:string"/>
      <xsd:element name="COMPOSER" type="xsd:string"/>
      <xsd:element name="PRODUCER" type="xsd:string"/>
      <xsd:element name="PUBLISHER" type="xsd:string"/>
      <xsd:element name="LENGTH" type="xsd:string"/>
      <xsd:element name="YEAR" type="xsd:string"/>
      <xsd:element name="ARTIST" type="xsd:string"/>
      <xsd:element name="PRICE" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

XML-Schema → XML-Dokument (2/2)

- Zuordnung eines Schemas, falls sich die Elemente in einen Namensraum befinden: Attribut **`xsi:schemaLocation`** im ersten Element hinzufügen, auf das das Schema angewendet wird (meistens: Root-Element)
- Wert von **`xsi:schemaLocation`** : Liste von Namensräume mit entsprechenden Schema-URIs

XML-Schema: Beispiel 2 (1/4)

```
<?xml version="1.0" encoding="UTF-8"?>
<SONG xmlns="http://ibiblio.org/xml/namespace/song"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation=
      "http://ibiblio.org/xml/namespace/song ex52.xsd">
  <TITLE>Yes I Am</TITLE>
  <COMPOSER>Melissa Etheridge</COMPOSER>
  <PRODUCER>Hugh Padgham</PRODUCER>
  <PUBLISHER>Island Records</PUBLISHER>
  <YEAR>1993</YEAR>
  <ARTIST>Melissa Etheridge</ARTIST>
</SONG>
```

XML-Schema: Beispiel 2 (2/4)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace=
  "http://ibiblio.org/xml/namespace/song"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:lied="http://ibiblio.org/xml/namespace/song"
  elementFormDefault="qualified">
  <xsd:complexType name="SongType">
    <xsd:sequence>
     <xsd:element name="TITLE" type="xsd:string"/>
     <xsd:element name="COMPOSER" type="xsd:string"/>
     <xsd:element name="PRODUCER" type="xsd:string"/>
     <xsd:element name="PUBLISHER" type="xsd:string"/>
     <xsd:element name="YEAR" type="xsd:gYear"/>
     <xsd:element name="ARTIST" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="SONG" type="lied:SongType"/>
</xsd:schema>
```

XML-Schema mit Namensräumen

- Starttag des XML-Schemas mit neuen Attributen:

```
<xsd:schema targetNamespace=
"http://ibiblio.org/xml/namespace/song"
xmlns="http://ibiblio.org/xml/namespace/song"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
```

- **`targetNamespace`**: URI des Namensraums der XML-Anwendung, welcher durch diesen Schema beschrieben ist
- **`elementFormDefault`**: Gibt an, ob sich die beschriebenen (lokalen) Elemente in einem **`targetNamespace`**-Namensraum befinden (**`qualified`**) oder nicht (**`unqualified`**)
- Globale Elemente sind immer im **`targetNamespace`**-Namensraum

Validierung von XML-Schemas mit Namensräumen

- Die URI (und nicht der Präfix) des Namensraums wird überprüft
  - Konsequenz: Das Dokument bleibt gültig, wenn die Präfixe geändert werden

XML-Schema: Beispiel 2 (3/4)

```
<?xml version="1.0" encoding="UTF-8"?>
<chanson:SONG xmlns:chanson=
"http://ibiblio.org/xml/namespace/song" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
"http://ibiblio.org/xml/namespace/song ex53.xsd">
  <chanson:TITLE>Yes I Am</chanson:TITLE>
  <chanson:COMPOSER>Melissa Etheridge</chanson:COMPOSER>
  <chanson:PRODUCER>Hugh Padgham</chanson:PRODUCER>
  <chanson:PUBLISHER>Island Records</chanson:PUBLISHER>
  <chanson:YEAR>1993</chanson:YEAR>
  <chanson:ARTIST>Melissa Etheridge</chanson:ARTIST>
</chanson:SONG>
```

XML-Schema: Beispiel 2 (4/4)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace=
"http://ibiblio.org/xml/namespace/song"
xmlns="http://ibiblio.org/xml/namespace/song"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xsd:element name="SONG" type="SongType"/>
  <xsd:complexType name="SongType">
    <xsd:sequence>
      <xsd:element name="TITLE" type="xsd:string"/>
      <xsd:element name="COMPOSER" type="xsd:string"/>
      <xsd:element name="PRODUCER" type="xsd:string"/>
      <xsd:element name="PUBLISHER" type="xsd:string"/>
      <xsd:element name="YEAR" type="xsd:gYear"/>
      <xsd:element name="ARTIST" type="xsd:string" />
   </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Element-Deklaration

- Elemente werden mit einem **`xs:element`**-Element deklariert
- Beispiel:

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

Element-Deklarationen und Namensräume

- Global Elemente sind immer im **`targetNamespace`**-Namensraum
- Nicht-globale Elemente sind per Default im **`targetNamespace`**-Namensraum, wenn das **`elementFormDefault`**-Attribut des **`xs:schema`**-Elements den Wert **`qualified`** hat

`minOccurs`- und `maxOccurs`-Attribute

- Die Minimal- und Maximalanzahl von Instanzen eines Elements, die in einem Instanzdokument auftreten dürfen, sind durch die Attribute **`minOccurs`** und **`maxOccurs`** des **`xs:element`**-Elements angegeben.
- Mögliche Werte des Attributs **`maxOccurs`** : natürliche Zahl oder **`unbounded`**
- Mögliche Werte des Attributs **`minOccurs`**: natürliche Zahl
- Beispiele

```
<xsd:element name="TITLE" type="xsd:string"
                  minOccurs="1" maxOccurs="1"/>
 <xsd:element name="ARTIST" type="xsd:string"
                  minOccurs="1" maxOccurs="unbounded"/>
 <xsd:element name="PRICE" type="xsd:string"
                  minOccurs="0" maxOccurs="1"/>
```

Typen

- Einfache Typen
  - Ohne abgeleitete Elemente
- Komplexe Typen
  - Enthalten Attribute und abgeleitete Elemente

Einfache Typen

- 44 vordefinierte einfache Typen, welche wie folgt unterteilt werden können
  - Numerische Typen (**`xs:float`**, **`xs:integer`**, **`xs:double`**, **`...`**)
  - Zeittypen (**`xs:timeInstant`**, **`xs:gYear`**, **`...`**)
  - XML-Typen (**`xs:ID`**, **`xs:ENTITY`**, **`xs:NOTATION`**, **`...`**)
  - Stringtypen (**`xs:string`**, **`xs:token`**,**`...`**)
  - Der Boolescher Typ (**`xs:boolean`**)
  - Der URI-Referent-Typ (**`xs:anyURI`**)

Neue einfache Type (1/2)

- Neue einfache Type können auf der Basis von anderen einfachen Typen mit Hilfe des **<xs:simpleType>**-Elements definiert werden
- Es gibt drei Methoden, um neue einfache Typen von existierenden einfachen Typen abzuleiten:
  - Einschränkung eines existierenden Typs mit einem **xs:restriction**-Child-Element
  - Vereinigung von existierenden Typen mit dem **xs:union**-Child-Element
  - Liste auf der Basis eines existierenden Typs mit dem **xs:list**-Child-Element

Neue einfache Type (2/2)

- Einschränkung

```
<xs:simpleType name="NewString">
  <xs:restriction base="xs:int"/>
 </xs:simpleType>
```

- Vereinigung

```
<xs:simpleType name="YearOrInteger">
  <xs:union memberTypes="xs:gYear xs:int"/>
 </xs:simpleType>
```

- Liste

```
<xs:simpleType name="listType">
   <xs:list itemType="xs:string"/>
</xs:simpleType>
```

Facette (1/2)

- Facette (engl. Facet): Aspekt eines möglichen Werts eines einfachen Datentyps
- Facetten eines einfachen Datentyps
  - whitespace
  - lenght (oder minLength, maxLength)
  - pattern
  - enumeration
  - maxInclusive, maxExclusive
  - minInclusive, minExclusive
  - totalDigits
  - fractionDigits

Facette (2/2)

- Jede Facette kann mit einem entsprechenden Child-Element im `xs:restriction`-Element deklariert werden

Facette: Beispiel 1

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
       <xs:element name="Kurzeichen" type="Abbreviation"/>
       <xs:simpleType name="Abbreviation">
              <xs:restriction base="xs:string">
                     <xs:maxLength value="8"/>
              </xs:restriction>
       </xs:simpleType>
</xs:schema>
```

schema

```
?xml version="1.0" encoding="UTF-8"?>
Kurzeichen xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="ex54.xsd">12345678</Kurzeichen>
```

gültig

```
?xml version="1.0" encoding="UTF-8"?>
Kurzeichen xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="ex54.xsd">
 12345678 </Kurzeichen>
```

ungültig

Facette: Beispiel 2

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Betrag" type="money"/>
  <xsd:simpleType name="money">
    <xsd:restriction base="xsd:string">
    <xsd:whiteSpace value="collapse"/>
    <xsd:pattern value="\p{Sc}\p{Nd}+(\.\p{Nd}\p{Nd})?"/>
   </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Betrag xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ex55.xsd"> € 123.56 </Betrag>
```

`xs:pattern` siehe E. Harold & al., *XML in a Nutshell*, s. 388-390

Komplexe Typen

- Ein komplexer Typ ist durch das **`xs:complexTyp`**-Element definiert

Komplexe Typen: Beispiel (1/2)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="SONG" type="SongType"/>
  <xsd:complexType name="SongType">
    <xsd:sequence>
      <xsd:element name="TITLE"     type="xsd:string"/>
      <xsd:element name="COMPOSER"  type="xsd:string"/>
      <xsd:element name="PRODUCER"  type="xsd:string"/>
      <xsd:element name="PUBLISHER" type="xsd:string"/>
      <xsd:element name="LENGTH"    type="xsd:string"/>
      <xsd:element name="YEAR"      type="xsd:string"/>
      <xsd:element name="ARTIST"    type="xsd:string"/>
      <xsd:element name="PRICE"     type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

(File: `song.xsd`)

Komplexe Typen: Beispiel (2/2)

```
<?xml version="1.0"?>
<SONG xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="song.xsd">
  <TITLE>Yes I Am</TITLE>
  <COMPOSER>Melissa Etheridge</COMPOSER>
  <PRODUCER>Hugh Padgham</PRODUCER>
  <PUBLISHER>Island Records</PUBLISHER>
  <LENGTH>4:24</LENGTH>
  <YEAR>1993</YEAR>
  <ARTIST>Melissa Etheridge</ARTIST>
  <PRICE>$1.25</PRICE>
</SONG>
```

Komplexe Typen

- Elemente von einem komplexen Typ haben Attribute und/oder Child-Elemente
- Attribute haben immer einen einfachen Typ
- Ein komplexer Typ, welcher durch einen Top-Element deklariert ist, muss eine Name haben und kann referenziert werden
- Ein komplexer Typ, der innerhalb einer Element Deklaration deklariert ist, braucht keine Name zu haben (anonymous Typ)
- Der komplexe Typ eines Elements beschreibt den Inhalt dieses Elements (siehe unten)

Anonymer Typ: Beispiel

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="SONG">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="TITLE"     type="xsd:string"/>
      <xsd:element name="COMPOSER"  type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
  </xsd:schema>
```

Elementsinhalt

Vier Möglichkeiten:

- Leerer Inhalt
- Einfacher Inhalt: nur Text (parsed character data)
- Gemischter Inhalt: Text und Elemente
- Beliebiger Inhalt

Leere Elemente

- Leere Elemente vermitteln Informationen durch Attribute oder durch ihre Position im Dokument (zB. `<br>` in XHTML)

Leeres Element: Beispiel

XML-Code:

```
<phoneNumber number= "055 432 34 67"/>
```

Entsprechende XML-Schema-Code

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

Einfacher Inhalt

- Einfacher Inhalt = Text ohne Elemente
- Elemente mit einfachem Inhalt sind
  - entweder von einem einfachen Typ
  - oder von einem komplexen Typ mit einem **`simpleContent`**-Element

`xs:simpleContent`-Element

- Das `xs:simpleContent`-Element deklariert, dass der Inhalt eines Elements einfach ist, d.h. nur aus Text und ohne Child-Elementen besteht

Einfacher Inhalt: Beispiel

XML-Code:

```
<?xml version="1.0" encoding="UTF-8"?>
  <Name xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="simpleElement.xsd">
      Jean Dupont
   </Name>
```

Entsprechende XML-Schema-Code

```
  <?xml version="1.0" encoding="UTF-8"?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="Name">
        <xs:complexType>
            <xs:simpleContent/>
        </xs:complexType>
      </xs:element>
    </xs:schema>
```

Gemischter Inhalt

- Der Inhalt eines Elements darf Text und untergeordnete Elemente enthalten, wenn
  - sein Typ komplex ist
  - das Attribut `mixed` des entsprechenden complexTyp-Elements den Wert `true` hat
- Die Position der untergeordneten Elemente ist durch die Elemente `xs:sequence`, `xs:choice` oder `xs:all` festgelegt
- `xs:sequence`, `xs:choice` und `xs:all` heissen **Gruppen**

Elementsgruppierungen

- `xs:all`- Gruppe : Alle Elemente treten höchstens einmal auf, wobei die Reihenfolge unwichtig ist
- `xs:choice`- Gruppe : Ein beliebiges Ellement aus der Gruppe darf auftreten
- `xs:sequence`-Gruppe: Alle Elemente treten genau einmal in der angegebenen Reihenfolge auf

Gemischter Inhalt: Beispiel (1/2)

- XML-Schema mit einer xs-sequence-Gruppierung

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="brief">
      <xs:complexType mixed="true">
         <xs:sequence>
            <xs:element name="datum"/>
            <xs:element name="betreff" type="xs:string"/>
            <xs:element name="unterschrift"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Gemischter Inhalt: Beispiel (2/2)

- Entsprechendes gültiges Dokument:

```
<?xml version="1.0" encoding="UTF-8"?>
  <brief xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="schema11.xsd">
  <datum>4. Oktober 2002</datum>
  <betreff>Besuch an der HSR</betreff>
    Lieber Peter
    Blablabla....
  <unterschrift/>
</brief>
```

Beliebiger Inhalt: `xs:any`-Element

- Das `xs-any`-Element gibt dem Parser an, wie die Validierung seines eigenen Inhalts durchgeführt werden soll

xs:any-Element: Beispiel

```
<xs:element name="notes">
  <xs:complexType>
    <xs:sequence>
    <xs:any namespace="http://www.w3.org/1999/xhtml"
        minOccurs="0" maxOccurs="unbounded"
        processContents="skip"></xs:any>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Attributdeklarationen

- Attribute sind mit dem **`xs:attribute`**-Element deklariert

Attributsdeklarationen: Beispiel (1/2)

Erstes Dokument (Name einer Person)

```
<?xml version="1.0"?>
 <Name xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance„
   xsi:noNamespaceSchemaLocation="schema3.xsd">
   Dupont
 </Name>
```

Zweites Dokument (Name einer Person mit ihrer Muttersprache als Attribut)

```
<?xml version="1.0"?>
 <Name xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance„
   xsi:noNamespaceSchemaLocation="schema3.xsd"
   language="french">
    Dupont
 </Name>
```

Frage: Wie sehen die entsprechenden Schemas aus?

Attributdeklarationen: Beispiel (2/2)

```
<?xml version="1.0"?>
<xsd:schema
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="Name" type="xsd:string"/>
</xsd:schema>

<?xml version="1.0"?>
<xsd:schema
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="Name" >
   <xsd:complexType>
     <xsd:simpleContent>
       <xsd:extension base="xsd:string">
         <xsd:attribute name="language"
                        type="xsd:string"></xsd:attribute>
       </xsd:extension>
     </xsd:simpleContent>
   </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

„Derived“ komplexe Typen

- Ableitung durch Erweiterung
  - Der neue Typ wird vom alten Typ durch Anhängen von zusätzlichen Elementen abgeleitet
  - Nur die neuen Elemente sind zu deklarieren
- Ableitung durch Einschränkung
  - Der neue Typ wird durch Auslassung von Teilen des alten Typs abgeleitet
  - Die Teile des alten Typs, die im neuen Typ übernommen werden, müssen neu deklariert werden

Beispiel: Ableitung durch Erweiterung

alt

```
<xs:complexType name="Adresse">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Strasse" type="xs:string"/>
    <xs:element name="Postleitzahl" type="xs:string"/>
    <xs:element name="Stadt" type="xs:string"/>
</xs:sequence>
```

neu

```
<xs:complexType name="LangAdresse">
  <xs:complexContent>
    <xs:extension base="Adresse">
      <xs:sequence>
        <xs:element name="TelefonNr" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Beispiel: Ableitung durch Einschränkung

alt

```
<xs:complexType name="Adresse">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Strasse" type="xs:string"/>
    <xs:element name="Postleitzahl" type="xs:string"/>
    <xs:element name="Stadt" type="xs:string"/>
</xs:sequence>
```

neu

```
<xs:complexType name="KurzAdresse">
  <xs:complexContent>
    <xs:restriction base="Adresse">
      <xs:sequence>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="Strasse" type="xs:string"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

Referenzen

- E. R. Harold, W.S. Scott: *XML in a Nutshell*, 2nd Edition, Kapitel 16 und 21