XML

e**X**tensible **M**arkup **L**anguage

Auszeichnungssprachen für Hyper-Text haben ihre Wurzel in SGML (Standard Generalized Markup Language):

Trennung von
Inhalt, Struktur und Darstellung
eines Dokumentes.

- Unter *Hyper-Text* versteht man die Verlinkung von unterschiedlichen Dokumenten, möglicherweise unterschiedlichen Formats.
- Eine *Auszeichnungssprache* macht die Struktur eines Dokumentes durch Verwendung von Tags explizit.
- Ein *Tag* hat einen Namen und möglicherweise Attribute; er annotiert (umfasst) einen Teil des Dokumentes.

- XML ist eine Metasprache zur Definition anwendungsspezifischer Auszeichnungssprachen; es existiert keine vorgegebene Menge von Tags.
- XML ersetzt nicht HTML; HTML ist eine spezielle Sprache gemäß XML (XHTML).
- Logische Struktur und Darstellung eines Dokumentes werden getrennt.
- XML erleichtert maschinelle Verarbeitung von Dokumenten, beispielsweise durch Software-Agenten.
- Internationalisierung (Unicode) und Plattformunabhängigkeit werden unterstützt.
- XML ist das von W3C standardisierte Datenaustauschformat des Internet.
- XML ist das Format der Zukunft für strukturierte Dokumente.

Beispiel: XML-Dokument.

```
<Land ID='f0215' Name='Germany' Einwohner='83536'
      Flche='356910' Autonummer='D'>
   <Provinz ID='f017524' Name='Baden Wurttemberg'
         Einwohner='10272' Fläche='35742'>
      <Stadt ID='f02623'>
         <Name>Karlsruhe</Name>
         <Einwohner Jahr='95'>277</Einwohner>
      </Stadt>
      <Stadt ID='f02852' >
         <Name>Freiburg im Breisgau</Name>
         <Einwohner Jahr='95'>198496</Einwohner>
      </Stadt>
   </Provinz>
</Land>
```

Element

- XML-Dokumente werden durch Tags strukturiert; die Verwendung eines Tags definiert ein *Element* des Dokuments.
- Ein Element mit Namen `ELEM` ist der Teil des Dokumentes, der durch seinen öffnenden Tag `<ELEM>` und schließenden Tag `</ELEM>` umfaßt ist. Der umfaßte Teil des Dokumentes ist der *Inhalt* des Tags.
- Ein Element kann selber Elemente enthalten; insbesondere kann ein Element mehrere Elemente mit demselben Tag enthalten.
- Elemente sind geordnet.
- Elemente können leer sein, d.h. keinen Inhalt haben. Kurznotation: `<ELEM/>`.

Beispiel:

```
<Land Name='Germany' >
   <Lage Kontinent='Europe' Prozent='100'/>
   <Grenze Land='Frankreich' Länge='300'/>
   <Grenze Land='Schweiz' Länge='100'/>
</Land>
```

Attribut

- Elemente können durch *Attribute* weiter beschrieben sein. Attribute werden innerhalb des öffnenden Tags in der Form `attr='value'` angegeben.
- Der Wert eines Attributes ist eine Zeichenkette vom Typ `CDATA`, `ID` oder `IDREF`, `IDREFS`.
- Attributwerte vom Typ `ID` sind eindeutig im betreffenden Dokument.
- Ein Attributwert vom Typ `IDREF` muß innerhalb des Dokumentes als Wert eines Attributes vom Typ `ID` auftreten. Analog ist ein Wert eines Attributs vom Typ `IDREFS` eine Liste von `ID`-Werten.
- Die Reihenfolge unter den Attributen eines Tags ist ohne Bedeutung.

Beispiel:

```
<Land Name='Germany' >
   <Lage Kontinent='Europe' />
</Land>
<Kontinent Name='Europe'/>
```

Ein XML-Processor ist ein Software-Modul, das XML-Dokumente liest und Anwendungen Zugriff zu Inhalt und Struktur des Dokumentes ermöglicht.

- Ein *Entity* bezeichnet eine Zeichenkette oder ein externes File. Innerhalb eines Dokumentes können *Entity-Referenzen* auftreten. Entity-Referenzen beginnen mit `'&'` und enden mit `';'`. Vordefinierte Entities existieren für <, >, &, ; und '. Entity-Referenzen werden im Dokument durch ihren Inhalt ersetzt. Im allgemeinen können beliebige Unicode-Zeichen als Entities verwendet werden.
- In '<![CDATA[' und ']]>' eingefaßte Zeichenketten werden vom XML-Processor unverändert übernommen.
- *Processing instructions* sind in '<?' . . . '?>' eingefaßte Anweisungen für den XML-Processor.
- *Kommentare* beginnen mit '<!-' und enden mit '->'; sie dürfen nicht die Sequenz '--' enthalten.
- Ein Dokument kann eine *Document Type Definition (DTD)* enthalten, in der die erlaubten Tag-Strukturen festgelegt sind.

wohl-geformte Dokumente

Ein XML-Dokument (ohne DTD) heißt *wohl-geformed*, wenn es den folgenden Bedingungen genügt:

- Das Dokument beginnt mit einer XML-Deklaration, z.B.
  `<?XML version='1.0' encoding='UTF-8' standalone='yes'?>`
  - Diese Deklaration ist der *Prolog* des Dokumentes.
  - `standalone` hat den Wert 'yes', sofern dem Dokument durch eine externe DTD oder ein Schema ein Typ zugeordnet ist und anderenfalls den Wert 'no'.
- Es existiert ein Wurzel-Element, das alle anderen Elemente enthält.
- Die Elemente sind passend genestet; ein öffnender Tag und sein schließender Tag haben dieselben umfassenden Tags.
- In einem gegebenen Tag darf jedes Attribut maximal einmal auftreten.

Für ein XML-Dokument können unterschiedliche und sogar überlappende Vokabulare (*Namesräume*) verwendet werden.

- Jeder XML-Tag besteht deshalb aus einer Namensraum-Angabe (*Namespace*) und einem lokalen Namen.
- Ein Namensraum wird durch einen URI repräsentiert; die URI ist hier kein Link, sondern lediglich ein eindeutiger Bezeichner.
- Es wird angenommen, dass ein XML-Processor aufgrund der Namensraum-Angaben weiß, wie er das Dokument zu verarbeiten hat.
- Ein Namensraum wird mittels dem Attribut `xmlns` festgelegt. Wird kein spezieller Prefix dem Namensraum zugeordnet, so ist er der Default-Namensraum.

Beispiel:

```
<?xml version='1.0'?>
   <Buch xmlns='urn:loc.gov:books'
         xmlns:isbn='urn:ISBN:0-840-65441-6'>
      <Titel>Alle Länder dieser Welt</Titel>
      <isbn:Nummer>1767643579</isbn:Nummer>
   </Buch>
```

Document Type Definition (DTD)

- Mittels einer DTD kann die Struktur der zulässigen Dokumente und die zulässigen Wertebereiche von Attributen festgelegt werden. Desweiteren können Defaultwerte für Attribute festgelegt und Entities definiert werden.
- Eine DTD kann als Teil eines Dokuments, oder unabhängig von diesem unter einer URL abgelegt sein.
- Ein Dokument, das konform zu seiner DTD ist, heißt *gültig (valid)*.
- XML verlangt, dass Dokumente wohl-geformt sind. Dokument können ohne zugehörige DTD existieren; Gültigkeit ist eine optionale Eigenschaft.
- Eine DTD ist formal eine Grammatik, deren Regeln die zulässigen Tag-Strukturen und Attribute festlegen.

Eine DTD wird deklariert wie folgt:

```
<!DOCTYPE aName [aDTDdeclaration]>
```

- `aName` ist der Name des äußersten Tags.
- `aDTDdeclaration` sind die Regeln der DTD.

In den Regeln werden Elemente mit ihren Attributen definiert. *Beispiel:*

```
<!DOCTYPE Mondial[
   <!ELEMENT Mondial (Land*, Kontinent*,
         (Meer*, Fluss*, Insel*)*)>
   <!ELEMENT Stadt (Name, ((Lage, Einwohner) |
         (Einwohner, Lage)))>
]>
```

Element-Typ-Deklaration:

```
<!ELEMENT elemName content>
```

`content` ist von einer der folgenden Formen:

- `EMPTY`: Elemente von diesem Typ haben keinen Inhalt.
- `ANY`: Es sind beliebiger Textinhalt, oder beliebige anderweitig definierte Elemente erlaubt.
- *mixed*: Der Inhalt besteht aus einer (parsed) Zeichenkette oder einer beliebigen Folge von angegebenen Elementen:

  ```
  (#PCDATA | elemName1 | elemName2 | ... )
  ```

- *children*: Der Inhalt ist definiert durch einen regulären Ausdruck über ElementNamen unter Verwendung der folgenden Operatoren:
  - *choice:* `( ... | ... | ...)`
  - *sequence:* (..., ..., ...)

    wobei die Häufigkeit des Vorkommens eingeschränkt wird durch
  - *optional:* ... ?
  - *zero or more:* ... $*$
  - *one or more:* ... +

Attribut-List-Deklaration:

```
<!ATTLIST elemName attName1 attType1 default1 attName2 attType2
                                        default2 ...>
```

`attType` ist von einer der folgenden Formen (Auswahl):

- `CDATA`: Es sind beliebige Zeichenketten erlaubt.
- Eine Aufzählung der erlaubten Werte: `( val1 | val2 | ...)`
- `ID, IDREF, IDREFS`.

`default` ist von einer der folgenden Formen:

- `#REQUIRED`: Das Attribut muß explizit aufgeführt werden.
- `#IMPLIED`: Das Attribut ist optional; es ist kein Default-Wert vorgesehen.
- `val`: Ist das Attribut nicht aufgeführt, dann wird dieser Wert angenommen.
- `#FIXED val`: Das Attribut muß immer den angegebenen Wert haben.

```
<!DOCTYPE Mondial[
   <!ELEMENT Mondial (Land*, Kontinent*,
         (Meer*, Fluss*, Insel*)*)>
   <!ELEMENT Stadt (Name, ((Lage, Einwohner) |
         (Einwohner, Lage)))>
   <!ATTLIST Land
      Name CDATA #REQUIRED
      Autonummer ID #IMPLIED
      HStadt IDREF #IMPLIED
      Mitglied IDREFS 'EU'>
]>
```

mit DTD’s verbundene Probleme:

- Separate Syntax; kann als Teil des Dokumentes angegeben werden.
- Unzureichendes Typkonzept für Zeichenketten und Attribute.
- Element-Definitionen sind global und berücksichtigen somit nicht den Kontext.
- Innerhalb regulärer Ausdrücke können keine Zeichenketten auftreten.
- Keine Unterstützung für Namensräume.
- Keine Unterstützung für Evolution, Erweiterung und Vererbung.
- Zu simpler ID-Mechanismus (kein Konzept vergleichbar zur referentiellen Integrität); dies nur über Attributen.
- Keine Defaults für Elemente; keine *wild-cards* für Elemente und Attribute.
- Reihenfolgeabhängigkeit der Elemente. Unabhängigkeit nur sehr umständlich definierbar.

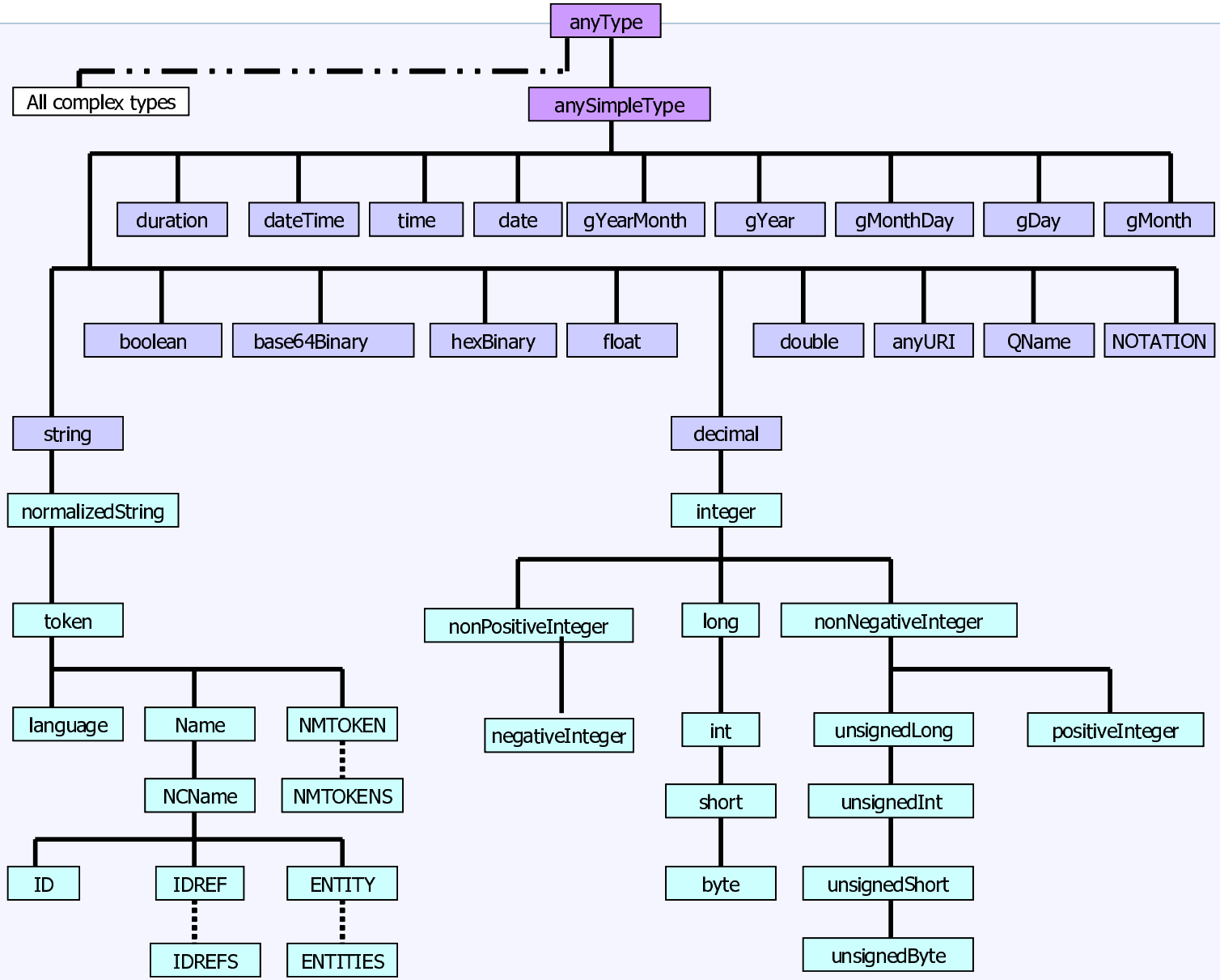
XML-Schema

- Es werden Typdefinitionen zur Verwendung in XML-Dokumenten bereitgestellt,
  - gegen die die Dokumente dann validiert werden können,
  - und die zum anderen Informationen explizit machen, wie beispielsweise Default-Werte.
- Typen werden definiert und dann über Deklarationen Namen zugeordnet.
  - Beschränkungen für Elemente,
  - Beschränkungen für Attribute.

Erweiterbares Typkonzept:

- eine Anzahl von Datentypen ist vordefiniert (*built-in primitive/derived types*),
- auf dieser Basis können neue Typen definiert werden,
- es wird zwischen einfachen Typen
  - ein Typ ohne Attribute und Kindelemente,

  und komplexen Typen
  - ein Typ mit Attributen und Inhaltsmodell,
- unterschieden.


anyType
All complex types
anySimpleType
duration
dateTime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth
boolean
base64Binary
hexBinary
float
double
anyURI
QName
NOTATION
string
normalizedString
token
language
Name
NMTOKEN
NCName
NMTOKENS
ID
IDREF
ENTITY
IDREFS
ENTITIES
decimal
integer
nonPositiveInteger
negativeInteger
long
int
short
byte
nonNegativeInteger
unsignedLong
unsignedInt
unsignedShort
unsignedByte
positiveInteger

- Komponenten eines einfachen Datentyps:
  - eine Menge voneinander verschiedener Werte (*Wertebereich, value space*),
  - eine Menge von Repräsentationen (*Repräsentationsraum, lexical space*),
  - eine Menge von weiteren Eigenschaften (*Aspekten, facets*).
- Zu jedem Wert des Wertebereichs existiert mindestens ein Literal im Repräsentationsraum. Beispielsweise existieren mehr Literale für Werte von Zeittypen, oder auch vom Typ `float` (100, 1.0E2).
- *fundamentale* Aspekte eines Datentypes wie Ordnung und Kardinalität können nicht verändert werden,
- Wertebereich und Repräsentationsraum können durch Aspekte eingeschränkt werden,
  - *beschränkende* Aspekte des Wertebereichs sind Unter-/Obergrenzen der Werte, Längenbeschränkungen und Aufzählungen der erlaubten Werte.
  - Ein Repräsentationsraum kann mittels eines regulären Ausdrucks eingeschränkt werden.
- Der Aspekt *whiteSpace* steuert, wie die Zeichen Leerzeichen, Tabulator, CR, LF bei der Abbildung auf den Repräsentationsraum behandelt werden sollen.

- Erzeugung von Datentypen aus gegebenen Datentypen mittels
  - Einschränkung (*restriction*),
  - Erweiterung (*extension*),
- Erzeugung neuer einfacher Typen aus einfachen Typen mittels
  - Listenbildung (*list*),
  - Vereinigung (*union*).
- Erzeugung von *Inhaltsmodellen* komplexer Typen mittels
  - Reihung (*sequence*) von Elementen in der vorgegebenen Reihenfolge,
  - Auswahl (*choice*) eines Elementes unter den angegebenen,
  - Konjunktion (*all*) in der Weise, dass jedes der angegebenen Elemente maximal einmal in beliebiger Reihenfolge auftritt.
  - Formal ist dies immer eine Einschränkung oder Erweiterung eines anderen Typs; dies kann der vordefinierte Typ `anyTyp` sein.
- Substitution group

Beispiele

```
<xs:simpleType name='Prozent'>
  <xs:restriction base='xs:number'>
    <xs:fractionDigits value='2'/>
    <xs:minInclusive value='0.00'/>
    <xs:maxInclusive value='100.00'/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name='Autonummer'>
  <xs:restriction base='xs:string'>
    <xs:pattern value='[A-Z]+....'/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name='AlleAutonummern'>
  <xs:list itemType='Autonummer'/>
</xs:simpleType>

<xs:simpleType name='LandCode'>
  <xs:restriction base='xs:NMTOKEN'>
    <xs:enumeration value='D'/>
    <xs:enumeration value='A'/>
    <xs:enumeration value='CH'/>
    <xs:enumeration value='F'/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:complexType name='Einwohner'>
  <xs:simpleContent>
    <xs:extension base='xs:positiveInteger'/>
      <xs:attribute name='Jahr' type='gYear'
                    use='required'/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name='Land'>
  <xs:complexContent>
    <xs:restriction base='xs:anyType'>
      <xs:sequence>
        <xs:element name="LCode" type="xs:string"/>
        <xs:element name="Name" type="xs:string"/>
        <xs:element ref="Provinz" minOccurs="0"
                    maxOccurs="unbounded"/>
        <xs:element ref="Lage" minOccurs="0"
                    maxOccurs="unbounded"/>
        <xs:element ref="Mitglied" minOccurs="0"
                    maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:restricition>
  </xs:complexType>
</xs:complexType>
```

Deklarationen

- *Globale* Deklarationen stehen auf oberster Ebene des Dokumentes. Sie sind mittels `ref` von anderen Stellen aus referenzierbar.
- Alle anderen Deklarationen sind *lokal* und nur in ihrem jeweiligen Kontext sichtbar.
- Deklarationen können auf außerhalb definierte Typen verweisen, oder eine direkte (*anonyme*) Typdeklaration enthalten.
- Offene Schemata:
  - Es können *wildcards* für Elemente (`any`) und Attribute (`anyAttribute`) verwendet werden, wobei diese durch Angabe eines Namensraums eingeschränkt werden können.
  - Die Verbindlichkeit einer Element-Deklaration kann gesteuert werden durch Angabe von `processContent='strict'` (verbindlich), `processContent='lax'` (zu beachten, falls vorhanden) und `processContent='skip'` (soll ignoriert werden).

einschränkende Bedingungen

- Eindeutigkeit von Werten.
  - Eindeutigkeit kann für Element- und Attributinhalte, oder auch für Kombinationen hiervon, verlangt werden. `unique` verlangt Eindeutigkeit für vorhandene Werte, während bei `key` der Wert vorhanden und eindeutig sein muss.
  - Eindeutigkeit bezieht sich relativ zu dem direkt übergeordneten Element (Kontext), in dem die Bedingung definiert ist.
  - Mittels `selector` wird die zun betrachtende Knotenmenge des Dokumentes relativ zum Kontext festgelegt.
  - Mittels `field` die Kombination der konkreten Werte relativ zum Selektor.
  - `selector` und `field` werden durch eingeschränkte XPath-Ausdrücke festgelegt.

- Referenzbeziehung `keyref` in Analogie zur referentiellen Integrität relationaler Datenbanken.
  - Mit `refer` wird der Bezug zu einem eindeutigen Kriterium festgelegt.
  - Mittels `selector` und `field` wird der Fremdschlüssel festgelegt, analog zu oben.
  - Der Kontext der Fremdschlüsseldefinition muss den Kontext der betreffenden Schlüsseldefinition enthalten.

Beispiele

```
<xs:element name = "Mondial">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref = "Land" minOccurs = "0"
                               maxOccurs = "unbounded"/>
      <xs:element name = "Provinz">
        <xs:complexType>
          <xs:sequence>
            <xs:element name = "Name" type = "xs:string"/>
            <xs:element name = "Land" type = "xs:string"/>
            <xs:element name = "Stadt" minOccurs = "0"
                                 maxOccurs = "unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:key name = "PrimaryKeyForLand">
    <xs:selector xpath = "Land"/>
    <xs:field xpath = "LCode"/>
  </xs:key>
  <xs:keyref name = "ForeignKeyForLand" refer = "PrimaryKeyForLand">
    <xs:selector xpath = "Provinz"/>
    <xs:field xpath = "Land"/>
  </xs:key>
</xs:element>
```

weitere Aspekte:

- Namenssräume,
- Ersetzungsgruppen:

```
<xs:complexType name="Land">
  <xs:sequence>
    <xs:element name = "LCode" type = "xs:string"/>
    <xs:element name = "Name" type = "xs:string"/>
        ...
  </xs:sequence>
</xs:complexType>

<xs:element name="carCode" type="xs:string"
  substitutionGroup="LCode"/>
```

  Anstatt `LCode` kann auch `carCode` verwendet werden.

- Instanzbezogene Konzepte, z.B. `nillable = 'true'` in einer Elementdeklaration erlaubt den Attributwert `xsi:nil ='true'`,
- phyisches Schema, z.B. mittels `annotation`-Element.

Schwächen von XML-Schema:

- ungeordneter Inhalt nut schwer beschreibbar, da `all` mit vielen Einschränkungen versehen ist,
- keine allgemeinen Integritätsbedingungen,
- `key` und `keyref` haben nur innerhalb eines Dokumentes eine Bedeutung,
- `keyref` und `list` sind unzureichend abgestimmt,
- ...
- XML-Schema ist in seiner Gesamtheit sehr komplex und schwer durchschaubar.

trotz allem ist XML-Schema ein Fortschritt gegenüber einer DTD.