Kapitel 10
XML Schema

ANFORDERUNGEN UND ZIELE

- man benötigt eine Schemabeschreibungssprache
  (Verwendung ist optional- XML ist selbstbeschreibend)
- DTD: Erbe von SGML; Datenbank-typische Schemainformationen teilweise nicht unterstützt (Datentypen [alles ist CDATA/PCDATA], Kardinalitäten); dafür Ordnung, Iteration.
- DTD: Syntax ist nicht XML

⇒ Besserer Formalismus, um Schema darzustellen

- XML-Syntax → leicht zu parsen
- detailliertere Informationen als in der DTD
- Datentypen mit abgeleiteten Typen, Constraints etc

XML SCHEMA: IDEEN

- keine grundlegend neuen Konzepte (etwa im Gegensatz zur Definition des objektorientierten Modells), sondern ...
- Kombination der Fähigkeiten vieler Schemasprachen:
  - Datentypen-Ansätze aus dem Datenbankbereich (SQL)
  - Idee der komplexen Objekttypen/klassen aus dem OO-Bereich
  - Strukturtypen aus dem Bereich der Baumgrammatiken (z.B. DTD)
- Neu gegenüber DTDs: Unterschied zwischen Elementtypen und Elementen: <country>-Elemente sind vom Typ countryType.

⇒ sehr komplex

XML Schema: Aufbau

Dank XML-Syntax und “geschwätzigem” Formalismus eine sehr detaillierte, systematische Herangehensweise an Typdefinitionen und -ableitungen

- einige, wenige vorgegebene primitive, atomare Datentypen
- davon werden durch *Restriktion* weitere einfache Datentypen (*simple types*) abgeleitet,
- und durch *Erweiterung* strukturierte Typen (*complex types*) abgeleitet.
- andere (*complex types*) werden von einem ganz allgemeinen *anyType* (vgl. Klasse *object* in OO) abgeleitet,
- und dann Elemente damit deklariert.

XML Schema: Der Standard

Die XML Schema Recommendation (Mai 2001) besteht aus 2 Teilen:

- Part 2: “Datatypes”
  - Definition von einfachen Datentypen (*simple types*)
    (haben keine Attribute und keinen Elementinhalt; werden nur als Text Contents und für Attributwerte verwendet)
- Part 1 “Structures”:
  - Definition von strukturierten Datentypen (*complex types*)
    (mit Subelementen und Attributen; werden als Elementtypen verwendet)
    * Namen/Typen der Subelemente und Attribute
    * Ordnung der Subelemente
  - Definition von Elementen unter Verwendung der complex types.
- viele Syntaxdefinitionen
- Part 0: “Primer” (`http://www.w3.org/TR/xmlschema-0/`) hier werden die Konzepte erklärt.

UMGANG MIT XML SCHEMA

- Konzepte und Ideen verstehen (Primer, Vorlesung)
- Part 1 (Structures) ist der wichtigere Teil
  (die Datentypen aus Part 2 benötigt man nur zur Verfeinerung)
- in der Praxis anwenden
- Syntax ggf. in den W3C-Dokumenten nachschlagen
- Erfahrungen entwickeln

XML SCHEMA-DOKUMENTE

Ein XML-Schema-Dokument besteht aus

- einer Präambel und
- einer Menge von Definitionen und Deklarationen

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  content
</xs:schema>

content umfaßt die folgenen Arten von *Schemakomponenten*:

- Datentypdefinitionen (simple types und complex types)
- Attributdeklarationen
- Elementdeklarationen
- Kleinkram

10.1 Datentypen

Datentypen werden als 3-Tupel gesehen:

- Wertebereich (Menge, Kardinalität, Gleichheits- und Ordnungsbegriffe),
- lexikalische Repräsentation durch *Literale* (z.B. 100 auch als 1.0E2 und 1.0e+2)
- Menge von Eigenschaften

Hierbei kann der Wertebereich auf verschiedene Weisen spezifiziert sein:

- extensional (Aufzählung)
- intensional (mittels Axiome)
- Einschränkung eines anderen Wertebereichs
- durch Konstruktion aus anderen Wertebereichen: Listen, Mengen ...

DATENTYPEN

Datentypen werden durch einen zugrundeliegenden Wertebereich sowie *Eigenschaften (facets)* in mehreren voneinander unabhängigen Dimensionen charakterisiert. Diese Eigenschaften beschreiben die Unterschiede zwischen Datentypen.

Die grundlegenden Eigenschaften (jeder Datentyp hat sie) sind

- Gleichheit,
- Ordnungsrelation,
- untere und obere Schranke,
- Kardinalität (endlich vs. abzählbar unendlich),
- Numerisch vs. non-numerisch.

DATENTYPEN

- primitive Datentypen (sie sind alle vordefiniert)
- generierte Datentypen (sie werden von anderen Datentypen abgeleitet). Dies kann durch Aggregatbildung oder durch Einschränkung des Wertebereichs geschehen.
- die Spezifikation von XML Schema enthält bereits einige vordefinierte Datentypen:
- Primitive vordefinierte Typen:
  - string (mit vielen Subtypen: token, NMTOKEN),
  - boolean (lexical Repr.: true, false),
  - float, double,
  - decimal (mit vielen Subtypen: integer etc.),
  - duration, time, dateTime, ...
  - base64binary, hexBinary
  - anyURI (Universal Resource Identifier).
- generierte vordefinierte Typen:
  - integer, [non]PositiveInteger, [non]NegativeInteger, [unsigned](long|short|byte)

DATENTYPEN

Darunter gibt es einige XML-spezifische Datentypen (Subtypen von string), die auf Basis des XML-Standards definiert sind. Sie sind nur als Attributtyp vorgesehen (atomic und list types):

- NMTOKEN (Einschränkung von string gemäß Definition von XML-Tokens).
- NMTOKENS durch Listenbildung von NMTOKEN abgeleitet,
- IDREF/IDREFS analog,
- Name: XML Names
- NCName: non-colonized names
- language: Language-Codes nach RFC 1766,

CONSTRAINING FACETS

Durch die Angabe von einschränkende Eigenschaften können weitere atomare Datentypen abgeleitet werden:

- bei Zeichenketten: length, minlength, maxlength, pattern (durch reguläre Ausdrücke);
- bei numerischen Datentypen: maxInclusive, minInclusive, maxExclusive, minExclusive,
- bei Listen: length, minLength, maxLength
- bei dezimalen Datentypen: totalDigits (Anzahl Stellen), fractionDigits (Anzahl Nachkommastellen);
- enumeration (Definition des Wertebereiches durch Aufzählung),

... erschöpfende Detailbeschreibung im W3C-XMLSchema-Working Draft.

ERZEUGUNG VON EINFACHEN DATENTYPEN

Einfache Datentypen können mit <simpleType> aus anderen abgeleitet werden:

Ableitung durch Einschränkung

Einschränkung eines Basistyps (d.h., Angabe von weiteren einschränkenden Eigenschaften):

```
<xs:simpleType name="name">
  < xs:restriction base="simple-type">
    facets
  </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="mondial:carcode">
  <xs:restriction base="xs:string">
    <xs:maxlength value="3"/>
  </xs:restriction>
</xs:simpleType>
```

Ableitung durch Einschränkung

Beispiel:

```
<xs:simpleType name="mondial:longitude"
   <xs:restriction base="xs:float">
      <xs:minExclusive value="-180"/>
      <xs:maxInclusive value="180"/>
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="mondial:latitude"
   <xs:restriction base="xs:float">
      <xs:minExclusive value="-90"/>
      <xs:maxInclusive value="90"/>
   </xs:restriction>
</xs:simpleType>
```

definiert zwei abgeleitete, einfache Datentypen.

Aufgabe 14
Definieren Sie einen simpleType, mit dem man die Lage eines Bereichs (z.B. eines Sees) (grob) beschreiben kann. □

Anmerkung: Verwendung von SimpleTypes

- als Attribute (Details später):

  ```
  <xs:attribute name="car_code" type="mondial:carcode"/>
  ```

  kann z.B. in

  ```
  <country car_code="D"> ... </country>
  ```

  verwendet werden.

- als Elemente (Details später):

  ```
  <xs:element name="longitude" type="mondial:longitude"/>
  ```

  kann z.B. in

  ```
  <longitude>48</longitude>
  ```

  verwendet werden.
  Erst wenn man hierzu noch Attribute haben will, muss man einen <complexType> definieren.

Ableitung durch Einschränkung: Enumeration

- Aufzählung der erlaubten Werte

Beispiel (aus XML Schema Primer)

```
<xs:simpleType name="USState">
  <xs:restriction base="xs:string">
    <xsd:enumeration value="AK"/>
    <xsd:enumeration value="AL"/>
    <xsd:enumeration value="AR"/>
    <!-- and so on ... -->
  </xsd:restriction>
</xsd:simpleType>
```

- so weit hat man etwa dieselben Datentypen, wie man sie auch in SQL als Attributtypen mit Hilfe von Integritätsbedingungen einschränken könnte.
- zusätzlich:
  - "mehrwertige" Listentypen (aber immer noch simple)
  - komplexe Typen

Ableitung durch Listenbildung

```
<xs:simpleType name="name">
   <xs:list itemType="simple-type">
      facets          <!-- optional -->
   </xs:list>
</xs:simpleType>
```

- *simpleType* muß ein non-list simple type sein
- auch hier können facets durch innere Subelemente angegeben werden

```
<xs:simpleType name="countrylist">
   <xs:list item_type="carcode"/>
</xs:simpleType>
```

- Analog Mengenvereinigung mit xs:union und @xs:memberTypes.

10.2 Komplexe Datentypen

Komplexe Datentypen können mit <complexType> aus anderen abgeleitet werden. Sie beschreiben

- ContentType: simpleContent oder complexContent (bereits definierter simpleType oder eigenes ContentModel)
- Attribute

Verschiedene Möglichkeiten:

- durch Erweiterung eines einfachen Datentyps (hinzufügen von Attributen, wodurch es explizit zum Elementtyp wird)
- durch Einschränkung eines anderen Datentyps (Einschränkung eines seiner Komponenten, oder seines Aufbaus),
- komplette Neudefinition (formal eine Beschränkung von base="xs:anyType")

Mit diesen Datentypen können später Elementtypen definiert werden.

KOMPLEXE DATENTYPEN

Verwendete "Stilmittel":

- Name,
  <xs:complexType name="*name*"> . . . </xs:complexType>
- Art des Inhalts (simple oder complex; mixed)
  <xs:simpleContent>
  <xs:complexContent [mixed="true"]>
- Ableitungsmethode (extension oder restriction),
  <xs:extension base="*typename*">
  <xs:restriction base="*typename*">
- Attributdeklarationen
  <xs:attribute name="*name*" type="*typename*"/>
- Aufbau des Contents-Model
  ... etwas komplexer

Komplexe Datentypen: Typ mit Attributen

Population: Text-contents und ein Attribut:

<population year="1997">130000</population>

- Man nimmt den simpleType des Text-Contents und erweitert ihn um ein Attribut:

```
<xs:complexType name=''population''>
  <xs:simpleContent>
    <xs:extension base=''nonNegativeInteger''>
      <xs:attribute name=''year'' type=''nonNegativeinteger''>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

- der complexType kann (muss aber nicht) denselben Namen haben, wie das später definierte Element.

Komplexe Datentypen: Leere Elementtypen

Border: nur zwei Attribute:

<border country="F" length="500"/>

- Man nimmt den Basistyp anyType (beliebiger complexContent und Attribute) und schränkt ihn ein:

```
<xs:complexType name=''border''>
  <xs:complexContent>
    <xs:restriction base=''anyType''>
      <xs:attribute name=''country'' type=''IDREF''>
      <xs:attribute name=''length'' type=''nonNegativeInteger''>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

Komplexe Datentypen: Beliebige Elementtypen

Neu definierte Elementtypen mit <complexContents> werden in der Regel als

```
<xs:complexType name=‘‘...’’>
  <xs:complexContent>
    <xs:restriction base=‘‘anyType’’>
```

erzeugt. Als Kurzform ist erlaubt, in diesem Fall, diese Spezifikationsmerkmale einfach wegzulassen:

```
<xs:complexType name=‘‘border’’>
   <xs:attribute name=‘‘country’’ type=‘‘IDREF’’/>
   <xs:attribute name=‘‘length’’ type=‘‘nonNegativeInteger’’/>
</xs:complexType>
```


Komplexe Datentypen: Beliebige Elementtypen

- Elementtypen mit komplexem Elementinhalt werden mit Hilfe strukturdefinierender Elemente (als *Model Groups* bezeichnet) erzeugt:
  - <xs:sequence> ... </xs:sequence>
  - <xs:choice> ... </xs:choice>
  - <xs:all> ... </xs:all>
    (mit Einschränkungen - nur top-level, keine Unterstrukturen erlaubt, maximale Kardinalität der Subelemente ist 1)
- darin werden die einzelnen Elemente angegeben:

  <xs:element name="*name*" type="*typename*">

Komplexe Datentypen: Beliebige Elementtypen

Beispiel

Definition des Typs der späteren <city>-Elemente:

```
<xs:complexType name=‘‘city’’>
  <xs:attribute name=‘‘country’’ type=‘‘IDREF’’/>
  <xs:sequence>
    <xs:element name=’name’ type=’string’/>
    <xs:element name=’population’ type=’population’/>
    <xs:element name=’longitude’ type=’longitude’/>
    <xs:element name=’latitude’ type=’latitude’/>
  </xs:sequence>
</xs:complexType>
```

... soweit die einzelnen Konzepte.

Aufgabe 15
Entwickeln Sie ein XML Schema für Mondial (oder für Ihre selbstdefinierte Anwendung). □

10.3 Zusammensetzen der Deklarationen

Weitere Attribute zu Attributdefinitionen

- default
- fixed
- use (optional, required, prohibited)
  default ist “optional”

Weitere Attribute zu Subelementdefinitionen

- default: wenn das Element mit leerem Inhalt im Dokument angegeben ist, wird der default-Wert als Contents eingesetzt.
  Ist das Element nicht angegeben, wird auch kein Default gesetzt.
- fixed: analog.
- minOccurs, maxOccurs: default 1.

Beispiele: später.

GLOBALE ATTRIBUT- UND ELEMENTDEFINITIONEN

... bis jetzt können beliebige Elementtypen definiert werden.

Zumindest für das root-Element benötigt man eine separate Element-Deklaration.

- <xs:attribute> und <xs:element>-Elemente können nicht nur innerhalb von <complexType>-Elementen auftreten, sondern auch global.
- dort dürfen sie aber keine use, maxOccurs, oder minOccurs-Spezifikation enthalten
- Solche globalen Deklarationen können dann auch in Typdefinitionen durch ref benutzt werden.
  Dort stehen dann auch use, maxOccurs und minOccurs.
- vor allem sinnvoll, wenn derselbe Elementtyp mehrfach genutzt wird.

BEISPIEL

```
<xs:element name=‘‘city’’ type=‘‘city’’/>
<xs:element name=‘‘name’’ type=‘‘string’’/>
<xs:attribute name=‘‘car_code’’ type=‘‘carcode’’/>

<xs:complexType name=‘‘country’’>
  <xs:sequence>
    <attribute ref=‘‘car_code’’ use=‘‘required’’/>
    <element ref=‘‘name’’/>
    <element ref=‘‘city’’ ref=‘‘name’’ maxOccurs=‘‘unbounded’’/>
    <element ref=‘‘border’’ minOccurs=‘‘0’’ maxOccurs=‘‘unbounded’’/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name=‘‘mondial’’>
  <xs:sequence>
    <element name=‘‘country’’ ref=‘‘country’’ maxOccurs=‘‘unbounded’’/>
    :
  </xs:sequence>
</xs:complexType>
```

ANONYME, LOKALE TYPDEFINITIONEN

- Anstatt
  <xs:element name="*name*" type="*typename*"/>
  <xs:attribute name="*name*" type="*typename*"/>
  können auch anonyme, lokale Typdefinitionen im Inhalt solcher Elemente angegeben werden:

```
<xs:complexType name=''city''>
  <xs:attribute name=''country'' type=''IDREF''/>
  <xs:sequence>
    <xs:element name='name' type='string'/>
    <xs:element name='population'>
      <xs:simpleContent>
        <xs:extension base=''nonNegativeInteger''>
          <xs:attribute name=''year'' type=''nonNegativeinteger''>
        </xs:extension>
      </xs:simpleContent>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```


LOKALE DEKLARATIONEN

- <complexType>-Deklarationen definieren lokale *Symbol spaces*, d.h. man kann denselben Attribut/Elementnamen in verschiedenen komplexen Datentypen mit unterschiedlichen Ergebnis-Datentypen vergeben (dies war in DTDs nicht möglich; vgl. country/population und city/population-Elemente)

```
<complexType name="countrypop"> ... </complexType>

<complexType name="citypop"> ... </complexType>

<complexType name="countryType">
   :
   <element name="population" type="countrypop"/>
</complexType>

<complexType name="cityType">
   :
   <element name="population" type="citypop"/>
</complexType>
```

ATTRIBUTGRUPPEN

Mehrfach verwendete Attributgruppen können benannt und definiert und dann verwendet werden:

<xs:attributeGroup name="*groupname*">
attributedefs
</xs:attributeGroup>

<complexType name="*name*" ...>
:
<xs:attributeGroup ref="*groupname*"/>
</complexType>

- kann auch geschachtelt werden

CONTENT MODEL GROUPS

Analog können auch Teile des Content Models vordefiniert werden:

<xs:group name="*groupname*">
modelgroupdef
</group>

<complexType name="*name*" ...>
:
<xs:group ref="*groupname*/>
</complexType>

Aufgabe 16
Verwenden Sie in Ihrem XML Schema für Mondial

- eine Attributgruppe für (country, province) bei city, lake, mountain etc.
- eine contents model group für (longitude, latitude) □

10.4 Integritätsbedingungen

XML Schema unterstützt drei weitere Integritätsbedingungen (*Identity constraints*):

- unique, key, keyref

die *sehr* starke Ähnlichkeit mit den entsprechenden SQL-Konzepten haben:

- einen Namen,
- einen *Selektor*: XPath-Ausdruck, z.B. //city, der die Menge der Elemente beschreibt unter denen die Bedingung gilt,
- eine Liste von Feldern (relativ zu den Ergebnis des Selektors), über die die Bedingung eine Aussage macht
- (bei keyref:) der Name einer key-Definition, die die referenzierten Schlüssel beschreibt.

Ausdruckskräftiger als ID/IDREF:

- mehrere Felder,
- kann nicht nur Attribute, sondern auch (textuellen) Elementinhalt umfassen.

INTEGRITÄTSBEDINGUNGEN

```
<unique name="..." >
    <selector xpath="xpath-expr"/>
    <field xpath="xpath-expr"/>
    :
    <field xpath="xpath-expr"/>
</unique>

<key name="name" >
    <selector xpath="xpath-expr"/>
    <field xpath="xpath-expr"/>
    :
    <field xpath="xpath-expr"/>
</key>

<keyref name="..."  refer="name" >
    <selector xpath="xpath-expr"/>
    <field xpath="xpath-expr"/>
    :
    <field xpath="xpath-expr"/>
</keyref>
```

INTEGRITÄTSBEDINGUNGEN

Beispiel:

```
<element name="country">
  <complexType>    <!-- lokaler Typ, ohne Name -->
    <attribute name="car_code" type="NMTOKEN"/>
    :
  </complexType>
</element>

<element name="mondial" type="mondialtype">
  <complexType>
   <element ref="country" maxOccurs="*">
   <element ref="organization" maxOccurs="*">
  </complexType>
  <key name="countrykey">
    <selector xpath=''//country''/>
    <field xpath=''@car_code''/>
  </key>
</element>
```


INTEGRITÄTSBEDINGUNGEN

Beispiel (Fortsetzung):

```
<element name="organization">
  <complexType>
     <element name="member" maxOccurs="*">
       <complexType name="membertype">
         <attribute name="type_of_membership"
                    type="NMTOKEN">
         <attribute name="country" type="NMTOKEN">
       </complexType>
     </element>
     <attribute name="name" type="string">
  </complexType>
  <keyref name="org-to-member-ref" refer="countrykey">
    <selector xpath=''.//member''/>
    <field xpath=''@country''/>
  </keyref>
</element>
```

AUFGABE

Definieren Sie sie referentiellen Integritätsbedingungen zu dem folgenden Fragment von Mondial:

```
<element name="country">
  <attribute name="car_code" type="NMTOKEN"/>
  <attribute name="capital"  type="string" />
  <key ... />
  <keyref .../>
</element>

<element name="city">
  <attribute name="country" type="string"/>
  <element   name="name"       type="string"/>
  <key ... />
  <keyref .../>
</element>

<country car_code="D" capital="Berlin"/>
<city in_country="D">
  <name>Berlin</name>
</city>
```