Chapter 10
XML Schema

10.1 Motivation

- Database area: schema description
  - cf. SQL datatypes, table schemata
  - constraints
- Programming languages: typing – *real* typing – means: theory
  - every expression (query, variable etc) can be assigned with a type
  - structural induction
  - static typechecking for queries/programs/updates
  - validation of resulting structures wrt. target DTD/Schema

XML Query Formal Semantics: Overview

Every (query) expression is assigned with a semantics

Static Semantics

Given a static environment, an expression is of a certain type:
(static env.: namespace decl, typedefs, type decls. of variables)

- $statEnv \vdash Expr : Type$

Dynamic Semantics

Given a dynamic environment, an expression yields a certain result:
(dynamic env.: context node, size+position in context, variable bindings)

- $dynEnv \vdash Expr \Rightarrow Value$
  (equivalent to "classical" notation: $[[Expr]]_{dynEnv} = Value$)

... both defined by structural induction.
(for short example: show 2.1.5 and "if" in 4.10 of W3C XQFS document)

XML Query Data Types

... by examples:

```
define type coordinates {  -- in any order
    element longitude of type xs:float &
    element latitude of type xs:float}

define type city {  -- sequence
    attribute country of type xs:string,
    attribute province of type xs:string?, -- optional
    element name of type xs:string,
    element population of type { -- anonymous
           attribute year of type xs:decimal,
           xs:string }
    element coordinates of type { -- anonymous
      element longitude of type xs:float &
      element latitude of type xs:float}    }
```

... similar to DTD expressions extended by primitive datatypes

XML Query Data Types (Cont'd)

XML type theory:

- operations on types (e.g. "union"): result type of a query that yields either a result of type a or type b
- derivation of new types by
  - additional constraints
  - additional content
- constraints: does the derived result type for some expression guarantee that some conditions hold?
- containment of types: is the derived result type for some expression covered by a certain target type?
  (static type checking of programs)
- can e.g. be applied for query and storage optimization, indexing etc.

Requirements on an XML Schema Language

- requirement: a schema description language for the user that is *based* on these types: (usage is optional – XML is self-describing)
- DTD: heritage of SGML; database-typical aspects are not completely supported (datatypes [everything is CDATA/PCDATA], cardinalities); but: order, iteration.
- DTD: syntax is not in XML.

⇒ better formalism for representing schema information

- XML syntax → easy to process
- more detailed information as in the DTD
- database world: datatypes with derived types, constraints etc.

XML Schema: Ideas

- no actually new concepts (in contrast to the definition of the object-oriented model), but ...
- combination of the power of previous schema languages:
  - datatype concepts from the database area (SQL)
  - idea of complex object types/classes from the OO area
  - structured types from the area of tree grammars (e.g. DTD)
- new in contrast to DTDs: difference between element types and elements: <country> elements are of the type countryType.

⇒ very complex

10.2 XML Schema: Design

Using XML syntax and a verbose formalism, XML Schema uses a very detailed and systematic approach to type definitions and -derivations.

- only a few primitive, atomic datatypes
- other *simple types* are derived from these by *restriction*,
- structured, *complex types* are derived by *extension*,
- other (*complex types*) are derived from a general *anyType* (cf. class *object* in OO),
- these types are then used for declaring elements.

XML SCHEMA: THE STANDARD

The XML Schema Recommendation (since May 2001) consists of 2 parts:

- Part 2: "Datatypes"
  - Definition of *simple types*:
    have no attributes and no element content; are used only for text content and as attribute values.
- Part 1 "Structures":
  - Definition of structured datatypes (*complex types*):
    with subelements and attributes; are used as element types.
    * names/types if the subelements and attributes
    * order of the subelements
  - Definition of elements using the complex types.
- many syntax definitions
- Part 0: "Primer" (`http://www.w3.org/TR/xmlschema-0/`) explains and motivates the concepts.

Usage of XML Schema

- understand concepts and ideas (XML Schema Primer, lecture)
- Part 1 (Structures) is the more important part
  (the datatypes from Part 2 are only needed for refinements)
- apply them in practice
- lookup for syntax details in the W3C documents
- make experiences

XML Schema Documents

An XML-Schema document consists of

- a preamble and
- a set of definitions and declarations

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

content uses the following kinds of *schema components*:

- datatype definitions (simple types and complex types)
- attribute declarations
- element declarations
- miscellaneous ...

10.3 Datatypes

Datatypes are seen as tripels:

- range (possible values, cardinality, equality and ordering),
- lexical representation by *literals* (e.g. 100 also as 1.0E2 and 1.0e+2)
- set of properties

The set of possible values can be specified in different ways:

- extensional (enumeration)
- intensional (by axioms)
- restriction of another domain
- construction from other domains (lists, sets ...)

DATATYPES

Datatypes are characterized by their domain and *properties (called facets)* in multiple independent “dimensions”. The facets describe the differences between datatypes.

The basic facets (present for each datatype) are

- equality,
- order relation,
- upper and lower bound,
- cardinality (finite vs. countable infinite),
- numerical vs. non-numerical.

DATATYPES

- primitive datatypes (predefined)
- generated datatypes (derived from other datatypes). This can happen by aggregation (lists) or restriction of another datatype.
- Primitive predefined types in XML Schema:
  - string (with many subtypes: token, NMTOKEN),
  - boolean (lexical repr.: true, false),
  - float, double,
  - decimal (with several subtypes: integer etc.),
  - duration, time, dateTime, ...
  - base64Binary, hexBinary
  - anyURI (Universal Resource Identifier).
- generated predefined types:
  - integer, [non]PositiveInteger, [non]NegativeInteger, [unsigned](long|short|byte)

XML-SPECIFIC DATATYPES

There are some XML-specific datatypes (subtypes of string) that are defined based on the basic XML recommendation. They are only used for attribute types (atomic and list types):

- NMTOKEN (restriction of string according to the definition of XML tokens),
- NMTOKENS derived from NMTOKEN by list construction,
- IDREF/IDREFS analogously,
- Name: XML Names,
- NCName: non-colonized names,
- language: language codes according to RFC 1766.

CONSTRAINING FACETS

By specifying constraining facets, further datatypes can be derived:

- for sequences of characters: length, minlength, maxlength, pattern (by regular expressions);
- for numerical datatypes: maxInclusive, minInclusive, maxExclusive, minExclusive,
- for lists: length, minLength, maxLength
- for decimal datatypes: totalDigits (number of digits), fractionDigits (number of positions after decimal point);
- enumeration (definition of the possible values by enumeration),

... for a description of all details, see the W3C-XMLSchema-Working Draft.

GENERATION OF SIMPLE DATATYPES

Simple datatypes can be derived as <simpleType> from others:

Derivation by Restriction

Restriction of a base type (i.e., specification of further restricting facets):

```
<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:minlength value="1"/>
      <xs:maxlength value="3"/>
    </xs:restriction>
</xs:simpleType>
```

Derivation by Restriction

Example:

```
<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:minInclusive value=“-90”/>
      <xs:maxInclusive value=“90”/>
    </xs:restriction>
</xs:simpleType>
```

defines two derived simple datatypes.

Remark: Usage of SimpleTypes

- as attributes (details later):

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

  can e.g. be used in

  ```
  <country car_code=“D”> ... </country>
  ```

- as elements (details later):

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

  can e.g. be used in

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

  Only if also attributes are required, a <complexType> must be defined.

Derivation by Restriction: Enumeration

- Enumeration of the allowed values.

Example (from XML Schema Primer)

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

- so far, this functionality is similar to what could be done in SQL by attribute types and integrity constraints.
- additionally:
  - "multi-valued" list types (but still simple types)
  - complex types

Derivation as List Types

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

- *simpleType* must be a non-list type,
- facets of the type can be defined by subelements

a datatype for a list of country codes:

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

Derivation as Union Types

- Analogously union of sets with xs:union and @xs:memberTypes.

Component of a data type for postal addresses for US suppliers: send e.g., to D 37075 Göttingen (car code), or CA 94065 Redwood (US State Code)

```
<xs:simpleType name="countrylist">
  <xs:union memberTypes="carcode USState"/>
</xs:simpleType>
```


10.4 Complex Datatypes

Complex datatypes can be derived from others by <complexType>. They describe

- ContentType: simpleContent or complexContent (an already defined simpleType or an own ContentModel),
- attributes.

Different possibilities:

- by extension from a simple datatype (adding attributes, making an element type out of a simple type)
- by restriction from another datatype (restriction of its components or its structure),
- completely new definition (formally, a restriction of base="xs:anyType")

With these datatypes, element types can be defined later.

Complex Datatypes

Complex datatypes are defined by the following properties:

- name,
  <xs:complexType name="*name*"> ... </xs:complexType>
- kind of content (simple or complex; mixed)
  <xs:simpleContent>
  <xs:complexContent [mixed="true"]>
- derivation method (extension or restriction),
  <xs:extension base="*typename*">
  <xs:restriction base="*typename*">
- attribute declarations
  <xs:attribute name="*name*" type="*typename*"/>
- structure of content model
  ... a bit more complex ...

Complex Datatypes: Type with Attributes

Population: text content and an attribute:

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

- take the simpleType for the text content and extend it with an attribute:

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

- the complexType can have (but does not necessary have) the same name, as the element to be defined later.

COMPLEX DATATYPES: EMPTY ELEMENT TYPES

Border: only two attributes:

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

- take the base type anyType (arbitrary complexContent and attributes) and restrict it:

```
<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>
```


COMPLEX DATATYPES: ARBITRARY ELEMENT TYPES

Newly defined element types with <complexContent> are usually defined by

```
<xs:complexType name="...">
  <xs:complexContent>
    <xs:restriction base="anyType">
      <!-- type definition -->
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

As an abbreviation, it is allowed to omit the specification of properties:

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

Complex Datatypes: Arbitrary Element Types

- element types with complex content use structure-defining elements (called *Model Groups*):
  - <xs:sequence> ... </xs:sequence>
  - <xs:choice> ... </xs:choice>
  - <xs:all> ... </xs:all>
    (with some restrictions - only top-level, no substructures allowed, maximal cardinality of the subelements is 1)
- inside, the allowed element types are specified:

  <xs:element name="*name*" type="*typename*"/>
- note: the XML Schema definition requires to list the content model specification before the attributes.

Complex Datatypes: Arbitrary Element Types

Example

Definition of the type of the <city> elements:

```
<xs:complexType name="city">
  <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:attribute name="country" type="IDREF"/>
</xs:complexType>
```

... so far, the concepts.

Exercise 10.1
Define an XML Schema for Mondial (or for your own application). □

10.5 Composing Declarations

Further Attributes of Attribute Definitions

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

Further Attributes of Subelement Definitions

- default: if the element is given in a document with empty content, then the default contents is inserted.
  In case that an element is not given at all, no default is used.
- fixed: analogous.
- minOccurs, maxOccurs: default 1.

Examples: later.

Global Attribute- and Element Definitions

... up to now, arbitrary element *types* can be defined.

At least, for the root element, a separate element declaration is needed.

- <xs:attribute> and <xs:element> elements can not only occur inside of <complexType> elements, but can also be global.
- as global declarations, they must not contain specifications of use, maxOccurs, or minOccurs.
- global declarations can then also be used in type definitions by `ref`.
  Then, they are allowed to have use, maxOccurs and minOccurs.
- especially useful if the same element type is used several times.

Example

```
<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>
    <xs:attribute ref="car_code" use="required"/>
    <xs:element ref="name"/>  -- maxOccurs and minOccurs default 1
    <xs:element ref="city" maxOccurs="unbounded"/>
    <xs:element ref="border" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="country" type="country"/>

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


Anonymous, Local Type Definitions

- Instead of
  <xs:element name="*name*" type="*typename*"/>
  <xs:attribute name="*name*" type="*typename*"/>
  anonymous, local type definitions in the content of such elements are allowed:

```
<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>
```

LOCAL DECLARATIONS

- <complexType> declarations define local *symbol spaces*, i.e., the same attribute/element names can be used in different complex datatypes with different specifications of result-datatypes (this is not possible in DTDs; cf. country/population and city/population elements)

Using global types:

<xs:complexType name="countrypop"> ... without @year ... </xs:complexType>

<xs:complexType name="citypop"> ... with @year ... </xs:complexType>

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

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


Local Declarations (Cont'd)

Using local "population" types:

```
<xs:complexType name="countryType">
   <xs:complexType name="pop"> ... without @year ... </xs:complexType>
      :
   <xs:element name="population" type="pop"/>
</xs:complexType>
```

```
<xs:complexType name="cityType">
   <xs:complexType name="pop"> ... with @year ... </xs:complexType>
      :
   <xs:element name="population" type="pop"/>
</xs:complexType>
```

ATTRIBUTE GROUPS

Groups of attributes that are used several times can be defined, named and then reused:

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

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

- group definitions can also be nested ...

CONTENT MODEL GROUPS

In the same way, parts of the content model can be predefined:

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

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

Exercise 10.2
Use the following group definitions in your MONDIAL schema:

- an attribute group for (country, province) in city, lake, mountain etc.
- a content model group for (longitude, latitude) □

10.6 Integrity Constraints

XML Schema supports three further kinds of integrity constraints (*identity constraints*):

- unique, key, keyref

that have *very* strong similarities with the corresponding SQL-concepts:

- a name,
- a *selector*: an XPath expression, e.g. //city, that describes the set of elements for which the condition is specified,
- a list of fields (relative to the result of the selectors), that are subject to the condition,
- (for keyref:) the name of a key definition that describes the corresponding referenced keys.

more expressive than ID/IDREF:

- not only document-wide keys
- multiple fields,
- can not only contain attributes, but also (textual) element content.

Integrity Constraints

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

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

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

Integrity Constraints

Example:

```
<xs:element name="country">
  <xs:complexType>    <!-- local Type, without Name -->
    <xs:attribute name="car_code" type="mondial:carcode"/>
    :
  </xs:complexType>
</xs:element>

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


Integrity Constraints

Example (continued):

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

EXERCISE

Define the referential integrity constraints to the following fragment of MONDIAL:

```
<xs:element name="country">
  <xs:attribute name="car_code" type="mondial:carcode"/>
  <xs:attribute name="capital"  type="string" />
  <xs:keyref> ... </xs:keyref>
</xs:element>
<xs:key> ... </xs:key>

<xs:element name="city">
  <xs:attribute name="country" type="mondial:carcode"/>
  <xs:element   name="name"       type="string"/>
  <xs:keyref> ... </xs:keyref>
</xs:element>
<xs:key> ... </xs:key>

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