Information Systems

WS 2005, JKU Linz
Course 10: XML Schema

Gábor Bodnár

URL: `http://www.risc.uni-linz.ac.at/education/courses/ws2005/is/`

Overview

- Schema declaration.
- Declaration of types, elements, attributes.

XML Schemas

XML schemas are templates for XML documents.

XML schemas are XML documents themselves that must follow strict syntactical rules.

They can specify what structures the modeled XML documents may have and impose restrictions on their contents.

There are also several software products available to validate schema definitions and XML documents against well defined schemas.

Properties of XML Schemas

- XML, schemas are valid XML documents themselves, so there is no additional parser necessary in XML applications.

- XML schemas provide a wide range of data types (integers, dates, strings, etc.) for element content and attribute values.

- Schemas allow the users to define custom data types, and support inheritance between types.

- Schemas also provide powerful features to express element groupings, and other properties of elements like uniqueness.

Schema Declaration

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"

  targetNamespace="http://big.company.com/product-info"

  xmlns:p="http://big.company.com/product-info"

  elementFormDefault="qualified"
  attributeFormDefault="qualified">

  <!--Schema definition comes here.-->

</xsd:schema>
```

Schema Declaration (continued)

Summary of the most important points:

- The root element must be `schema`, and it must be qualified to be in the namespace `http://www.w3.org/2001/XMLSchema`.
- The `schema` element can contain the attribute `targetNamespace` and other namespace declarations.
- The `schema` element can contain attributes `elementFormDefault` and `attributeFormDefault` to enforce/forbid qualifying local elements/attributes in the instance documents.

Referencing Schemas

An XML document that references a schema is called an instance of the schema.

To reference a schema in an instance XML document one has to:

- If a target namespace is declared in the schema, this also has to be declared in the instance,
- define the schema instance namespace, which is the predefined namespace `http://www.w3.org/2001/XMLSchema-instance`,
- reference the schema by setting the value of the `schemaLocation` attribute (belonging to the schema instance namespace) to the URI of the schema.

Example

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://big.company.com/product-info">
  <!--Schema definition comes here.-->
</xsd:schema>
```

Instance:

```
<p:list xmlns:p="http://big.company.com/product-info"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://big.company.com/schemas/product-info.xsd">

  <!--Some content.-->
</p:list>
```

Element Declarations

The structure of the instance documents of a schema is principally described by the structure of the schema itself.

For instance if we want to prescribe the following structure in instances

```
<e1>
  <e2>
    <e3> ... </e3>
  </e2>
  <e4> ... </e4>
</e1>
```

we will encapsulate the declarations of these elements using this structure in the schema.

Declaration of Simple Elements

Simple elements, can contain only character data, possibly in some prescribed format.

The following line appearing in a schema definition declares an element whose tag must be `simple` and whose content can be any character data.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://big.company.com/product-info">

  <xsd:element name="simple" type="xsd:string"/>

</xsd:schema>
```

Complex Element Structures

sequence This requires that the instance contains the elements that appear in the schema in the same order by default exactly once.

all This allows that elements in the definition group appear in any order by default exactly once.

choice This allows only one of the elements in the definition group to appear in an instance document inside the complex element.

minOccurs Specifies the minimal number of occurrences of the declared element in its parent element.

maxOccurs Specifies the maximal number of occurrences of the declared element in its parent element.

Declaration Complex Elements

Fixed structure:

```
<xsd:element name="product">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="id" type="xsd:unsignedInt"/>
      <xsd:element name="price" type="xsd:float"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Example: Complex Elements

An instance element of the previous schema fragment.

```
<product>
  <name>Disney Mouse Pad</name>
  <id>231069948</id>
  <price>3.49</price>
</product>
```

Declaration Complex Elements

Fixed content that can appear in any order.

```
<xsd:element name="product_extension">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="made_in" type="xsd:string"/>
      <xsd:element name="oem" type="xsd:boolean"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

Example: Complex Elements

Instance elements of the previous schema fragment.

```
<product_extension>
  <made_in>USA</made_in>
  <oem>false</oem>
</product_extension>

<product_extension>
  <oem>true</oem>
  <made_in>Japan</made_in>
</product_extension>
```

Declaration Complex Elements

Occurrence prescription.

```
<xsd:element name="product_sold_in">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="country" type="xsd:string"
                   minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Declaration of Elements with Mixed Content

Using the `mixed` attribute of the `complexType` schema element.

```
<xsd:element name="note">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element name="emp" type="xsd:string"
                   minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

An instance:

```
<note>The extra lecture will be <emp>tomorrow at
16:00</emp> in the lecture hall <emp>HS 13</emp>.</note>
```

Attribute Declarations

Attributes are declared similarly to child elements, but they always have simple content.

Attributes can be defined only inside complex element declarations.

The attribute declarations always come after the child element declarations.

In an attribute definition the `name` and `type` attributes will prescribe the corresponding qualities of the attribute being declared.

The `use` attribute controls requirements for the declared attribute and the `value` attribute can set default or fixed values for it.

Constraining Attributes with `use`

required The declared attribute must be explicitly defined every time its element occurs in the instance of the schema.

default The declared attribute is optional and if not given the `value` attribute specifies its value.

optional The declared attribute is optional, no default is given.

fixed The declared attribute is optional, but if it appears, its value has to be the one given in the `value` attribute.

prohibited The declared attribute must not appear in any occurrence of the corresponding element and it has no predefined value.

Example

We want to create a schema for the following kind of instances:

```
<product>
  <id>231069948</id>
  <price currency="EUR">3.49</price>
</product>
```

where the structure is fixed and the presence of the attribute of the `price` element is required.

Unfortunately the definition is a bit lengthy.

Example

```
<xsd:element name="product">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="id" type="xsd:unsignedInt"/>
      <xsd:element name="price">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:float">
              <xsd:attribute name="currency"
                type="xsd:string" use="required"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
     </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Using References

Global declarations are the children of the root element in a schema.

In local declarations we can refer to global ones via the `ref` attribute to use the global declaration to define the referencing element.

For instance assuming that the lengthy definition of the `price` element is available as a global declaration,

```
<xsd:element ref="price"/>
```

would suffice inside the declaration of the `product` element.

Type Declarations

XML schema provides more than forty built-in data types and the possibility to define new types from the already existing ones.

The previous examples already contained anonymous type definitions which we used only at the place of definitions:

```
<xsd:complexType> ... </xsd:complexType>
```

We can create named types by setting its `name` attribute.

```
<xsd:complexType name="mytype"> ... </xsd:complexType>
```

Then inside the scope of the above type definition we can use the named type in further element declarations.

```
<xsd:element name="myelement" type="mytype"/>
```

Example

```
<xsd:complexType name="price_type">
  <xsd:simpleContent>
    <xsd:extension base="xsd:float">
      <xsd:attribute name="currency"
        type="xsd:string" use="required"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:element name="product">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="id" type="xsd:unsignedInt"/>
      <xsd:element name="price" type="price_type"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Type Restriction

Built-in data types provide one or more “facets” through which one can create new types by restriction.

Example

Strings that match a given regular expression.

The new type contains strings format `nnn-nn-nnnn` where in place of an `n` any digit can stand (e.g. 323-90-0982).

```
<xsd:simpleType name="serial_number">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-\d{2}-\d{4}"/>
  </xsd:restriction>
</xsd:simpleType>
```

Type Extension

For complex data types one can also use extension to add new elements and attributes to an already existing type.

Example

```
<xsd:complexType name="extended_product_type">
  <xsd:complexContent>
    <xsd:extension base="product_type">
      <xsd:sequence>
        <xsd:element name="made_in" type="xsd:string"/>
        <xsd:element name="oem" type="xsd:boolean"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Prescribing Uniqueness

We can prescribe that elements or attributes have unique values within a given scope in an instance of the XML schema.

```
<xsd:element name="list">
  <xsd:complexType> <xsd:sequence>
      <xsd:element name="product">
        <xsd:complexType> <xsd:sequence>
            <xsd:element name="id" type="xsd:unsignedInt"/>
            <xsd:element name="price" type="xsd:float"/>
        </xsd:sequence> </xsd:complexType>
      </xsd:element>
  </xsd:sequence> </xsd:complexType>
  <xsd:unique name="unique_id_declaration">
    <xsd:selector xpath="product"/>
    <xsd:field xpath="id"/>
  </xsd:unique>
</xsd:element>
```

Keys and Keyrefs

Referential integrity can be enforced via the `key` and `keyref` schema elements. They are used similarly to the `unique` element.

```
<xsd:element name="attends">

  <!--Defining the student and course children.-->

  <xsd:key name="course_key">
    <selector xpath="course"/>
    <field xpath="cid"/>
  </xsd:key>
  <xsd:keyref name="course_ref" refer="course_key">
    <selector xpath="student"/>
    <field xpath="cid"/>
  </xsd:keyref>
</xsd:element>
```

Summary

- Schema declaration and referencing
- Declaration of elements
  - ⋆ Simple elements
  - ⋆ Complex elements (sequence, all, choice, . . .)
  - ⋆ Complex elements with mixed content
- Declaration of attributes (required, default, optional, fixed, prohibited)
- Type declarations (restrictions, extensions)
- Uniqueness, Keys and Keyrefs