ITB295/ITN295 XML
Lecture Notes on XML Schema Design

1 Introduction

1.1 Topics

In these notes, we discuss some of the issues surrounding the use of namespaces in XML Schema and in instance documents. In particular, we discuss:

- Namespaces and XML Schema
- Four general XML Schema design patterns
- The *substitution group*

References:

✓ `http://www.xfront.com/GlobalVersusLocal.html`

✓ `http://www.rpbourret.com/xml/NamespacesFAQ.htm`

✓ `http://www.medbiq.org/working\_groups/journal/JournalCEguidelines.pdf`

✓ `http://devresource.hp.com/drc/resources/xmlSchemaBestPractices.jsp`

✓ `http://www.xfront.com/BestPracticesHomepage.html`

✓ `http://www.xmlpatterns.com/`

2 Namespaces and XML Schema

2.1 Namespaces

```
<contract xmlns="http://www.flashlawyers.com">
 <party>
  <qut:dept xmlns:qut="http://www.qut.edu.au/schemas/legal/">
   <qut:deptname>Information Services</qut:deptname>
   <qut:phone>+61 7 3864 2111</qut:phone>
  </qut:dept>
 </party>
 <party xmlns:abc="http:www.abc.net.au/podline">
  <abc:deptname>Podcasting</abc:deptname>
  <abc:address>123 Hutton St</abc:address>
 </party>
</contract>
```

2.2 A reminder

- A namespace is a vocabulary of (element and attribute) names based on some established domain.
- With each namespace, there is an established URI (more commonly an URL) that identifies the space.
- XML names will no longer contain the colon (`:`) symbol.
- Such names are called NCNames (non-colonised names).
- A *qualified name* is an NCName optionally preceded by a prefix and a colon.
- The prefix is associated with the URI that identifies the name space.

2.3 Global vs local

- Any element declaration that is the child of the `schema` root element is said to be global in scope.
- Elements declared below that level are said to be "local" in scope.
- Global element names must be unique.
- Global elements can be referenced both from within the current schema and from other schemas.
- Essentially, global components are reusable components.

Reference:

✓ `http://www-128.ibm.com/developerworks/xml/library/x-tiplocdec.html`

2.4 elementFormDefault qualified

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

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

```
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string"/>
      <xsd:element name="Age" type="xsd:integer"/>
      <xsd:element name="Sex">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="M"/>
            <xsd:enumeration value="F"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

The Russian Doll

2.5 Comments

- The schema element contains a `targetNamespace` attribute.
- All global schema components will belong to this namespace.
- Any instance document validated against this schema *must* employ a namespace.
- The schema element also contains an attribute `formElemenDefault`.
- This has been set to "`qualified`".
- As a consequence, every *local element* must also be qualified.

2.6 Two instance documents

Consider this instance document:

```
<Person xmlns="http://www.person.schema.org">
 <Name>Frank</Name>
 <Age>21</Age>
 <Sex>M</Sex>
</Person>
```

And the following:

```
<me:Person xmlns:me="http://www.person.schema.org">
 <me:Name>Frank</me:Name>
 <me:Age>21</me:Age>
 <me:Sex>M</me:Sex>
</me:Person>
```

2.7 Document validity

With regard to the schema, *both* documents are valid:

- The first document is valid because every element is in the default namespace.
- The second is valid because every element has its name qualified with an explicit prefix.

2.8 elementFormDefault unqualified

```
<xsd:schema
   targetNamespace="http://www.person.schema.org"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="unqualified">


<xsd:element name="Person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string"/>
      <xsd:element name="Age" type="xsd:integer"/>
      <xsd:element name="Sex">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="M"/>
            <xsd:enumeration value="F"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

2.9 Comments

- The schema element contains an attribute `formElemenDefault`.
- This has been set to "`unqualified`".
- The `Person` element has a global declaration in the schema used.
- The `Name`, `Age` and `Sex` elements have local declarations.

2.10 Two more instances

Now consider the following instance document:

```
<me:Person xmlns:me="http://www.person.schema.org">
 <Name>Frank</Name>
 <Age>21</Age>
 <Sex>M</Sex>
</me:Person>
```

And the following:

```
<Person xmlns="http://www.person.schema.org">
 <Name>Frank</Name>
 <Age>21</Age>
 <Sex>M</Sex>
</Person>
```

2.11 Document validity

- The first instance document is valid with regard to this schema:
  - The `Person` element is in the correct namespace.
  - The other three elements should not be and are not in any namespace.
- The second instance document is *invalid* with regard to this schema:
  - The `Person` element is in the correct namespace, but it is also the default namespace.
  - The other three elements are also in the default namespace, but they should be *unqualified*, that is, they should not be in *any* namespace at all.

3 Four design possibilities

3.1 Design criteria

Some criteria are:

- *Visibility*: to what extent are the components of a schema visible from outside the schema and from elsewhere in that schema.

  If we want to reuse a definition, can we?

- *Coupling*: to what extent are the components intertwined?

  If we want to change a definition, what is its impact?

3.2 The Russian Doll

The two schemas used to specify the `<Person/>` document have certain characteristics:

- There is only one global element – the document root.
- There are no named types: all types are anonymous.
- This style of specification is called the *Russian Doll* design.

3.3 The Salami Slice

In this alternative approach to writing XML Schema, the following approach is taken:

- Every element is defined at the global level.
- Compound elements make use of the `ref` attribute to reference their child elements.
- There are no named types: all types are anonymous.

Because all elements are global, there is no clearly defined root element.

3.4 Declaring simple global elements

Simple elements are introduced like this:

```
<xsd:element name="Name" type="xsd:string"/>
<xsd:element name="Age" type="xsd:integer"/>
<xsd:element name="Sex">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="M"/>
      <xsd:enumeration value="F"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

3.5 Declaring complex global elements

Elements with child content are introduced like this:

```
<xsd:element name="Person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Name"/>
      <xsd:element ref="Age" />
      <xsd:element ref="Sex" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

3.6 Design patterns

- The *Russion Doll* approach.
- The *Salami Slice* approach.
- The *Venetian Blind* approach.
- The *Garden of Eden* approach.

References:

✓ `http://www.xfront.com/GlobalVersusLocal.html`

✓ `http://developers.sun.com/prodtech/javatools/jsenterprise/nb\_enterprise\_pack/reference/techart/design\_patterns.html`

3.7 The Venetian Blind

To write an XML Schema following this approach:

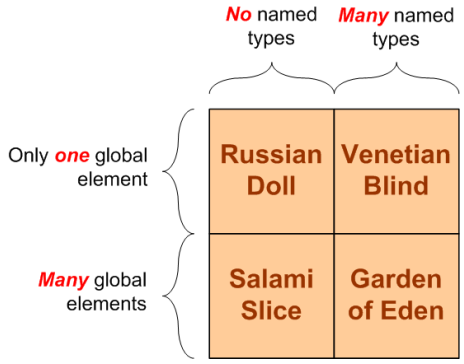
1. Write a simple type definition for each attribute.
2. Write a simple type definition for each simple element that has no attributes.
3. Write a complex type definition for each simple element that has attributes.
4. Write a complex type definition for all other elements *except* the root element.
5. Write a global element declaration for the root element only.

3.8 The Garden of Eden

To write an XML Schema following this approach:

1. Write a simple type definition for each attribute.
2. Write a simple type definition for each simple element that has no attributes.
3. Write a complex type definition for each simple element that has attributes.
4. Write a complex type definition for all other elements *except* the root element.
5. Write a global element definition for *every* element.

3.9 The four patterns



3.10 Best practice

- The Venetian Blind design is the one to choose where your schemas require the flexibility to turn namespace exposure on or off with a simple switch, and where component reuse is important.
- Where your task requires that you make available to instance document authors the option to use element substitution, then use the Salami Slice design.
- Where mimimizing size and coupling of components is of utmost concern then use the Russian Doll design.

Reference:

✓ `http://www.xfront.com/GlobalVersusLocal.html`

4 A worked example

4.1 Hot Gossip

Consider the following report:

```
The Hot Gossip Report
--------------------------------
Name   Contact

Ann    22 Strand Bvd, Copenhagen
Bill   3391 1615
Sue    8223 2555
Doug   3 Via Appia, Rome
--------------------------------
```

4.2 and the following encoding:

```
<HotGossip>
 <Gossip>
  <Name>Ann</Name>
  <Contact>
   <Address>22 Strand Bvd, Copenhagen</Address>
  </Contact>
 </Gossip>
 <Gossip>
  <Name>Bill</Name>
  <Contact>
   <PhoneNr>3391 1615</PhoneNr>
  </Contact>
 </Gossip>
 <Gossip>
  <Name>Sue</Name>
  <Contact>
   <PhoneNr>8223 2555</PhoneNr>
  </Contact>
 </Gossip>
 <Gossip>
  <Name>Doug</Name>
  <Contact>
   <Address>3 Via Appia, Rome</Address>
  </Contact>
 </Gossip>
</HotGossip>
```

4.3 The Salami Slice pattern

We will write the schema following the Salami Slice pattern:

- All elements are defined at the global level: `HotGossip`, `Gossip`, `Name`, `Contact`, `Address` and `PhoneNr`
- The compound elements `HotGossip`, `Gossip` and `Contact` make use of the `ref` attribute to reference their child elements.

4.4 Declaring simple global elements

Simple elements are introduced like this:

```
<xsd:element name="Name">
 <xsd:simpleType>
  <xsd:restriction base="xsd:string">
   <xsd:minLength value="1"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>

<xsd:element name="Address" type="xsd:string"/>
<xsd:element name="PhoneNr" type="xsd:string"/>
```

4.5 Declaring complex global elements

Elements with child content are introduced like this:

```
<xsd:element name="Contact">
 <xsd:complexType>
  <xsd:choice>
   <xsd:element ref="Address"/>
   <xsd:element ref="PhoneNr"/>
  </xsd:choice>
 </xsd:complexType>
</xsd:element>
```

4.6 'Gossip' and 'HotGossip'

And this:

```
<xsd:element name="Gossip">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="Name"/>
   <xsd:element ref="Contact"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<xsd:element name="HotGossip">
 <xsd:complexType>
  <xsd:sequence maxOccurs="unbounded">
   <xsd:element ref="Gossip"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

5 Substitution groups

5.1 Substitution groups

- In the worked example, only two forms of contact are allowed. But surely there are other forms by which we may get in touch with our friends.
- The use of the `choice` element to specify the content of the `<Contact/>` element makes it difficult to extend.
- Substitution groups provide a way around that.
- We provide a list of the allowable contact child elements – which could be stored separately.

5.2 The focus of substitution

The *head* of the group might be introduced in the following way:

```
<xsd:element name="Contact" type="xsd:anyType">
 <xsd:annotation>
  <xsd:documentation>
  Head of the Contact substitution group
  </xsd:documentation>
 </xsd:annotation>
</xsd:element>
```

- The type anyType means that any kind of value may be used for a contact.
- This is, in fact, the default value for the type attribute.
- The annotation and documentation elements may be used to document any element declaration.

5.3 The allowable substitutions

A list of all the permissible substitutions will be introduced:

```
<xsd:element name="Address" type="xsd:string"
     substitutionGroup="Contact"/>
<xsd:element name="PhoneNr" type="xsd:string"
     substitutionGroup="Contact"/>
<xsd:element name="Mobile"  type="xsd:string"
    substitutionGroup="Contact"/>
<xsd:element name="Email"   type="xsd:anyURI"
     substitutionGroup="Contact"/>
```

5.4 Abstract head

The *head* of the group might be introduced in the following way:

```
<xsd:element name="Contact" type="xsd:anyType"
             abstract="true"/>
```

- The element is said to be *abstract*, which means that there can never be an actual element called <Contact/>.
- Only substitutions based on this element may appear.

5.5 Hot Gossip – revisited

The document will now look like this:

```
<HotGossip>
 <Gossip>
  <Name>Ann</Name>
  <Address>22 Strand Bvd, Copenhagen</Address>
 </Gossip>
 <Gossip>
  <Name>Bill</Name>
  <PhoneNr>3391 1615</PhoneNr>
 </Gossip>
 <Gossip>
  <Name>Sue</Name>
  <PhoneNr>8223 2555</PhoneNr>
 </Gossip>
 <Gossip>
  <Name>Doug</Name>
  <Address>3 Via Appia, Rome</Address>
 </Gossip>
</HotGossip>
```

6 Conclusions

6.1 This week's topics

In these notes, we examined some of the issues surrounding the use of namespaces in XML Schema and in instance documents. In particular, we discussed four (yes, four) general XML Schema design patterns:

- Russian Doll
- Salami Slice
- Venetian Blind
- Garden of Eden

We also looked the use of substitution groups as a means of providing flexible choice.

6.2 Next week's topics

Next week, we will discuss XBRL (the eXtensible Business Reporting Language). In particular, we look at:

- The motivation behind XBRL.
- The most recent XBRL specification.
- The associated concepts: taxonomies and the discoverable taxonomy set.
- The XLink standard, and how it is used to extend XML Schema.
- An example.
- The IFRS-GP taxonomy.