# *Fundamentals of Web Programming* [a]

## *XML Schemas*

Teodor Rus

rus@cs.uiowa.edu

The University of Iowa, Department of Computer Science

# *Disadvantages of DTD-s*

- DTD-s are written in a syntax unrelated with XML so they cannot be validated by XML processors;
- It can be confusing to deal with two different notations while handling one document;
- There are only ten data types available with DTD-s, all of which are specified as text;
- DTD-s do not allow restrictions on the form of data content of a particular tag. **Example:** numeric content of an element can be integer, float, or a range of time.

# *XML Schema*

XML Schema provides a standard for XML document specification which can be parsed by XML parsers and provides more control over the data types than DTD-s.

- The contents of an XML element can be anyone of the 44 different data types supported;
- The user can define new types using constraints on the existing types;
- DTD-s can be automatically mapped into XML Schemas.

# *Schema Fundamentals*

A schema is similar to a class definition; an XML document that conforms to the structure defined by a schema is similar to an object of the class.

- A schema specifies the structure of its instance XML documents;
- A schema specifies the data type of every element and attribute in its instances XML documents;

Note: Namespaces in XML schemas are represented by unique URI-s. That is, they start with author's Web site address.

**Example:** the prefix

`http://cs.uccs.edu/`

is used in the textbook.

# *Facts*

1. XML schemas use a tag set from a namespace that defines a schema of schemas. This namespace is `http://www.w3.org/2001/XMLSchema`.
2. Some elements in this namespace are `schema, element, attribute, sequence, string`;
3. Schema of schemas namespace needs to be declared in each XML schema using the declaration `xmlns:xsd = "http://www.w3.org/2001/XMLSchema"`.

# *Defining a Schema*

Each schema has `schema` as its root element which have the following three attributes:

1. The namespace for schema of schemas using the declaration

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

2. The namespace defined by the schema using the declaration:

   `targetNamespace = "URI/schemaName"`

   where URI is the author Web address, as in:

   `targetNamespace = "http://cs.uiowa.edu/~rus/myFirstSchema"`;

3. If the elements and attributes that are not defined directly in the `schema` element (they are nested inside top-level element) are to be included in the target namespace the schema's attribute `elementFormDefault` must be set to `qualified` using the declaration `elementFormDefault = "qualified"`.

# *Observation*

The default namespace (i.e. the source of unprefixed names in the schema) is defined by the declaration

```
xmlns = "URI/~rus/schemaName"
```

as in

```
xmlns = "http://cs.uiowa.edu/~rus/myFirstSchem".
```

# *Example Schema Definition*

1. The default namespace is the tag set defined by the new schema:

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

2. The default namespace is the XMLSchema tag set:

```
<schema
   xmlns = "http://www.w3.org/2001/XMLSchema"
   targetNamespace = "http://cs.uccs.edu/planeSchema"
   xmlns:plane = "http://www.w3.org/2001/planeSchema"
   elementFormDefault = "qualified" >
```

**Note:** all names created by this schema must be prefixed by `plane` both in schema definition and in its instantiations.

# *Defining a Schema Instance*

An instance of a schema must include declarations of NS-s it uses in its root element:

1. First, the default NS is the NS defined by its schema. Using `plane` as root element this is obtained by the declaration:

   ```
   <plane xmlns = "http://cs.uccs.edu/planeSchema" ...  >
   ```

2. Second, the standard NS for XMLSchema instances declared by:

   xmlns:xsi = "http://www/w3.org/2001/XMLSchema-instance";

3. Third, the file name of the schema where the default namespace is defined. This is done using `schemaLocation` attribute which takes two values:

   (a) namespace of the schema, and

   (b) the file name of the schema:
   xsi:schemaLocation = "http://cs.uccs.edu/planesSchema planes.xsd"

# *Example*

The root element name in the instance of `plane.xsd` schema is:

```
<planes
   xmlns = "http://cs.uccs.edu/planeSchema"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://cs.uccs.edu/planeSchema planes.xsd" >
```

# XML Schema Data Types

1. **Simple data types:** used for elements whose content is restricted to strings. A simple type cannot have attributes or include nested element. Predefined data types are simple data types;
2. **Complex data types:** used for elements that can have attributes and include other data types as child element.

**Note:** XML Schema defines 44 data types, 19 of which are primitive and 25 of which are derived.

All XML schema data types are defined/commented/illustrated in `http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/`

# *Facts*

1. Primitive data types include: `string, Boolean, float, time, anyURI`;
2. Predefined derived types include `byte, long, decimal, unsignedInt, positiveInteger, NMTOKEN`;
3. User defined data types are defined by restrictions on existing types, called *base types*;
4. Constraints in derived types are given in terms of *facets* of base type. For example, integer primitive type has eight facets: `totalDigits, maxInclusive, maxExclusive, minInclusive, minExclusive, pattern, enumeration, whitespace`.

# *Observation*

The list of predefined types are at:

http://www.w3.org/TR/xmlschema-2/#built-in-datatypes

# *Local and Global Types*

- Both simple and complex types can be *named* and anonymous. An anonymous type cannot be used outside of the element in which it is declared.
- A local declaration in an XML schema is one that appear inside an element that is a child of the `schema` element.
- A local type is one that is declared in a grandchild of `schema` element and is visible only in that element.
- A global declaration is one that appear as a child of the `schema` element. Global elements are visible in the whole schema in which they are declared.

# *Simple Types*

- Elements are defined in an XML schema using `element` tag from XMLSchema.

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

- An instance of schema in which `engine` element is defined by:

```
<engine> inline size cilinder fuel injected </engine>
```

- An element can be given a default value using `default` attribute:

```
<xsd:element name = "engine" type = "xsd:string" defaut = "77" />
```

- Elements can have constant values if declared with:

```
<xsd:element name = "plane"
             type = "xsd:string"
             fixed = "single-wing" />
```

# User Defined Type

- A simple user-defined type is described in a `simpleType` element using facets inside the `restriction` element which give the base type name.
- **Example 1:** a user defined type using strings of fewer that 11 characters:

```
<xsd:simpleType name = "firstName" >
     <xsd:restriction base = "xsd:string">
          <xsd:maxLength value = "10" />
   </xsd:restriction>
</xsd:simpleType>
```

# *Example 2*

A user defined type using `precision` facet:

```
<xsd:simpleType name = "phoneNumber">
     <xsd:restriction base = "sxd:decimal">
          <xsd:precision value = "7" />
     </xsd:restriction>
</xsd:simpleType>
```

# *Complex Types*

Complex types are defined using the `complexType` tag.

- The `sequence` element is used to define an ordered group of elements:

```
<xsd:complexType name = "sports_car">
  <xsd:sequence>
     <xsd:element name = "make" type = "xsd:string" />
     <xsd:element name = "model" type = "xsd:string" />
     <xsd:element name = "engine" type = "xsd:string" />
     <xsd:element name = "year" type = "xsd:decimal" />
  </xsd:sequence>
</xsd:complexType>
```

# *More Complex Types*

A complex type whose element are an unordered group is defined in an `all` element.

- Elements in `all` and `sequence` groups can include attributes to specify the number of occurrences;
- These attributes are `minOccurs` (with possible values non-negative integers including 0) and `maxOccurs` (with possible values non-negative integers plus `unbounded`.

# Using minOccurs, maxOccurs

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!-- planes.xsd: a simple schema for planes.xml -->
<xsd:schema
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://cs.uccs.edu/planesSchema"
  xmlns = "http://cs.uccs.edu/planesSchema"
  elementFormDefault = "qualified">

<xsd:element name = "planes">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name = "make" type = "xsd:string"
                   minOccurs = "1" maxOccurs = "unbounded" />
    </xsd:all>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

# *XML Instance of* `planes.xsd`

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!-- planes1.xml
     An XML document instantiated from plane.xsd schema -->
<planes
  xmlns = "http://cs.uccs.edu/planesSchema"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://cs.uccs.edu/planesSchema planes.xsd">
    <make> Cessna </make>
    <make> Piper  </make>
    <make> Beechcraft </make>
</planes>
```

# *Defining Types by Reference*

If we want the `year` element in the `sports_car` element to be a derived type, the derived type could be defined by another global element which then can be referred to in the `sports_car` element.

**Example:**

```
<xsd:element name = "year">
  <xsd:simpleType>
    <xsd:restriction base = "xsd:decimal">
      <xsd:minInclusive value = "1900" />
      <xsd:maxInclusive value = "2008" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

# *Referring to "year"*

A reference to a defined type can be made using the attribute `ref`.

```
<xsd:complexType name = "sports_car">
  <xsd:sequence>
     <xsd:element name = "make" type = "xsd:string" />
     <xsd:element name = "model" type = "xsd:string" />
     <xsd:element name = "engine" type = "xsd:string" />
     <xsd:element ref = "year" type = "xsd:decimal" />
  </xsd:sequence>
</xsd:complexType>
```

# *Validating Instances of Schemas*

- Developing a schema is of limited value unless there is some mechanical way to determine whether a given XML document conforms to that schema;
- An XML Schema Validator (`xsv`) was developed by S. Thompson and R. Tobin, University of Edinburgh, Scotland.
- If schema and its instance are available on the Web, `xsv` can be used to validate it.
- The Web site of `xsv` is:

  `http://www.w3.org/XML/Schema#XSV`.

**Note:** There are many more XML schema validators available now. You can find them using `google`.

# *Development Methodology*

There are three methods used to develop XML schemas:

1. Following the structure of XML document;
2. Defining first all elements and attributes and then referring them through `ref`;
3. Using named types, i.e., naming the simpleType and complexType elements, and then point to them through the type attribute of the element.

Further we illustrate this methods for XML schema development by developing XML schema for the XML document called `shiporder.xml`

*shiporder.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<shiporder orderid="889923"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
</shiporder>
```

# *XML Schema for* `shiporder.xml`

- The file name of the XML schema for `shiporder.xml` is `shiporder.xsd`.
- Methodology: follow the structure of XML document and define each element as we find it.
- The standard declaration of XML schema is:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>
```

# *Now define* `shiporder` *element*

Note `shiporder` has an attribute and contains other elements. Therefore we use an ordered complex type.

```
<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto" type="ship2Type"/>
      <xs:element name="item" type="itemType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

# *Define next element*

`orderperson` element does not contain any attribute or element. Hence we use a simple type:

```
<xs:element name="orderperson" type="xs:string"/>
```

Next element is `shipto` which contains other elements. Therefore we need a complex type to define it.

# *Define* `shipto` *element*

```
<xs:element name="shipto">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="address" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Now we can define the `item` element, which contain other element, therefore we need again a complex type.

# *Define* `item` *element*

```
<xs:element name="item" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="note" type="xs:string" minOccurs="0"/>
      <xs:element name="quantity" type="xs:positiveInteger"/>
      <xs:element name="price" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

**Note:** there are no limits for the number of items one would like to ship.

Therefore the attribute `maxOccurs` is set to "unbounded".

# *Defining the attributes*

The attribute declarations must always come last. There is only one attribute to define, namely the `orderid` which is an attribute of the element `shiporder`.

```
<xs:attribute name="orderid" type="xs:string" use="required"/>
```

The complete listing of the XML schema file `shiporder.xsd` follows.

## shiporder.xsd

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
```

## shiporder.xsd *continuation*

```
      <xs:element name="item" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="note" type="xs:string" minOccurs="0"/>
            <xs:element name="quantity" type="xs:positiveInteger"/>
            <xs:element name="price" type="xs:decimal"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="orderid" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
</xs:schema>
```

# *Alternative methodology*

Methodology: define all elements and attribute first and then refer to them using the `ref` attribute.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- definition of simple elements -->
<xs:element name="orderperson" type="xs:string"/>
<xs:element name="name" type="xs:string"/>
<xs:element name="address" type="xs:string"/>
<xs:element name="city" type="xs:string"/>
<xs:element name="country" type="xs:string"/>
<xs:element name="title" type="xs:string"/>
<xs:element name="note" type="xs:string"/>
<xs:element name="quantity" type="xs:positiveInteger"/>
<xs:element name="price" type="xs:decimal"/>
```

# *Continuation*

```
<!-- definition of attributes -->
<xs:attribute name="orderid" type="xs:string"/>

<!-- definition of complex elements -->
<xs:element name="shipto">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name"/>
      <xs:element ref="address"/>
      <xs:element ref="city"/>
      <xs:element ref="country"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

# More complex elements

```
<xs:element name="item">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="note" minOccurs="0"/>
      <xs:element ref="quantity"/>
      <xs:element ref="price"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

# More complex element

```
<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="orderperson"/>
      <xs:element ref="shipto"/>
      <xs:element ref="item" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="orderid" use="required"/>
  </xs:complexType>
</xs:element>
</xs:schema>
```

# *Reusing element definitions*

- The third method used to define XML schemas allows us to reuse element definitions.
- This is accomplished by naming the simpleType and complexType elements, and then pointing to them through the type attribute of the element.

# *Third design of shiporder.xsd*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:simpleType name="stringtype">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="inttype">
  <xs:restriction base="xs:positiveInteger"/>
</xs:simpleType>
<xs:simpleType name="dectype">
  <xs:restriction base="xs:decimal"/>
</xs:simpleType>
<xs:simpleType name="orderidtype">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]{6}"/>
  </xs:restriction>
</xs:simpleType>
```

# Continuation

```
<xs:complexType name="shiptotype">
  <xs:sequence>
    <xs:element name="name" type="stringtype"/>
    <xs:element name="address" type="stringtype"/>
    <xs:element name="city" type="stringtype"/>
    <xs:element name="country" type="stringtype"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="itemtype">
  <xs:sequence>
    <xs:element name="title" type="stringtype"/>
    <xs:element name="note" type="stringtype" minOccurs="0"/>
    <xs:element name="quantity" type="inttype"/>
    <xs:element name="price" type="dectype"/>
  </xs:sequence>
</xs:complexType>
```

# *Continuation*

```
<xs:complexType name="shipordertype">
  <xs:sequence>
    <xs:element name="orderperson" type="stringtype"/>
    <xs:element name="shipto" type="shiptotype"/>
    <xs:element name="item" maxOccurs="unbounded" type="itemtype"/>
  </xs:sequence>
  <xs:attribute name="orderid" type="orderidtype" use="required"/>
</xs:complexType>

<xs:element name="shiporder" type="shipordertype"/>
</xs:schema>
```

# Facts

1. The restriction element indicates that the datatype is derived from a W3C XML Schema namespace datatype. So, the following fragment means that the value of the element or attribute must be a string value:

```
<xs:restriction base="xs:string">
```

2. The restriction element is more often used to apply restrictions to elements. Look at the following lines from the schema above:

```
<xs:simpleType name="orderidtype">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]{6}"/>
  </xs:restriction>
</xs:simpleType>
```

This indicates that the value of the element or attribute must be a string, it must be exactly six characters in a row, and those characters must numbers from 0 to 9.

# *Displaying XML Documents*

If an XML document is displayed without a style sheet that defines presentation styles for the documents tags, the displayed document will have no formatted content.

- Contemporary browsers include default style sheets that are used when no style sheet is specified for the XML document.
- If a style sheet fff.css is developed for a schema it needs to be provided in the XML document using the following directive:

```
<?xml-stylesheet type = "text/css" href = "fff.css" ?>
```

- The eXtensible Stylesheet Language (XSL) has been developed for this purpose.

# XSLT Processing

Figure 1 shows the tools used to transform an XML document into a XSL document using an XSLT processor and an XSLT document.

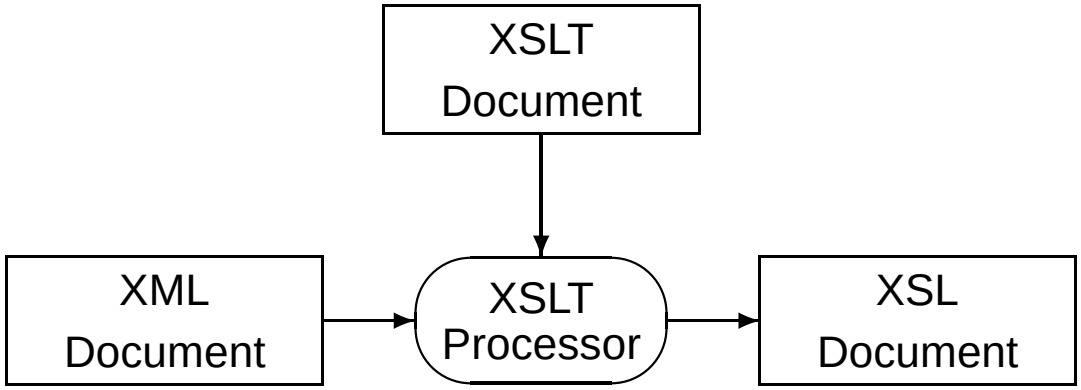


Figure 1: XSLT Processing

Use the following command on Linux:

```
xsltproc f1.xml f2.slt > f3slt
```