Schema Languages for XML

An **XML schema** is a description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type, above and beyond the basic syntactical constraints imposed by XML itself.

There are languages developed specifically to express XML schemas:

- DTD (Document Type Definition)
- XML **S**chema
- RELAX NG (REgular LAnguage for XML Next Generation)

Document Type Definition: DTD

- part of the original XML specification
- an XML document may have a DTD
- DTD is to describe the structure of XML documents.
- A DTD can be declared inline in your XML document, or as an external reference.
- An XML document is
  - well-formed: if tags are correctly nested and attributes of an element are unique.
  - valid: if it has a DTD and conforms to the DTD.
    - validation is useful in data exchange

A Simple DTD

- Consider an XML document consisting of an arbitrary number of person elements like

```
<db> <person> <name> Alan </name>
              <age> 42 </age>
              <email> agb@abc.com </email>
     </person>
     <person> .  .  .  </person>
     .  .  .
</db>
```

- A DTD for it might be

```
<!DOCTYPE persondb [
      <!ELEMENT db (person*)>
      <!ELEMENT person (name, age, email)>
      <!ELEMENT name (#PCDATA)>
      <!ELEMENT age  (#PCDATA)>
      <!ELEMENT email(#PCDATA)>
]>
```

More DTD

- Element person has a sequence of sub-elements, i.e. name, age, email.

  Order is important for child elements.

- * means 0 or more occurrences

  + means one or more occurrences

  ? means 0 or 1 occurrence

  | means alternation

- PCDATA means parsed character data.

  - Think of character data as the text found between the start tag and the end tag of an XML element.

  - Tags inside the text will be treated as markup and entities will be expanded.

More DTD (cont.)

- CDATA means character data.
  - CDATA is text that will not be parsed by a parser.
  - CDATA is used to describe data type of attributes.
  - Tags inside the text will not be treated as markup and entities will not be expanded.

- Entities are variables used to define shortcuts to common text.

  Syntax: <!ENTITY entity-name "entity-value">

  DTD Example:

  ```
  <!ENTITY writer1 "Donald Duck">
  <!ENTITY copyright1 "Copyright W3 School">
  ```

  XML data example:

  ```
  <author> &writer1; &copyright1; </author>
  ```

  will be expanded to

  ```
  <author> Donald Duck Copyright W3 School </author>
  ```

Note: An entity has three parts: an ampersand (&), an entity name, and a semicolon (;).

DTDs as Grammars

- A DTD is precisely a context-free grammar for the document.
- It imposes the child element order in an element.
- Grammars can be recursive, as in the following DTD describing binary trees:

```
<!ELEMENT node (leaf | (node, node))>
<!ELEMENT leaf (#PCDATA)>
```

Note: | means alternation.

- An example of such an XML document is

```
<node>
   <node>
        <node> <leaf> 1 </leaf> </node>
        <node> <leaf> 2 </leaf> </node>
   </node>
   <node>
      <leaf> 3 </leaf>
   </node>
</node>
```

More DTD: References and Attributes

- Attributes of elements are single valued attributes except for IDREFS attributes.
- the type ID declares that the particular attribute defines the element's unique identifier.
- the type IDREF means that the attribute's value is some other element's identifier.
- IDREFS means a list of identifiers.

E.g. A family tree specification

```
<!DOCTYPE family [
   <!ELEMENT family (person*)>
   <!ELEMENT person (name)>
   <!ELEMENT name (#PCDATA)>
   <!ATTLIST person  pid       ID      #REQUIRED
                     mother    IDREF   #IMPLIED
                     father    IDREF   #IMPLIED
                     children  IDREFS  #IMPLIED>
]>
```

Notes: **#REQUIRED** - The attribute value must be included in the element
#IMPLIED - The attribute does not have to be included, i.e. optional

More DTD: References and Attributes (cont…)

- A very simple XML element that conforms to this DTD is

```
<family>
    <person pid="John" children="Jane Jack">
        <name> John Doe </name>
    </person>
    <person pid="Mary" children="Jane Jack">
        <name> Mary Smith </name>
    </person>
    <person pid="Jane" mother="Mary" father="John">
        <name> Jane Doe </name>
    </person>
    <person pid="Jack" mother="Mary" father="John">
        <name> Jack Doe </name>
    </person>
</family>
```

Note: The parent-child relationships are duplicated.

More DTD: References and Attributes (cont…)

- E.g. Geography information specification (of a country)

```
<!DOCTYPE geography [
    <!ELEMENT geography (state+, city+>
    <!ELEMENT state (capital, cities-in*)>
        <!ATTLIST state scode  ID     #REQUIRED
                        sname  cdata #REQUIRED>
    <!ELEMENT capital EMPTY>
        <!ATTLIST capital   ccode IDREF  #REQUIRED>
    <!ELEMENT cities-in EMPTY>
        <!ATTLIST cities-in ccode IDREF  #REQUIRED>
    <!ELEMENT city (state-of)>
        <!ATTLIST city  ccode ID     #REQUIRED
                        cname cdata  #REQUIRED>
    <!ELEMENT state-of EMPTY>
        <!ATTLIST state-of  scode IDREF  #REQUIRED>
]>
```

Note: **EMPTY** means the element is empty, the element only has attributes.

Limitations of DTDs as schemas

- impose unwanted constraints on order of child elements.
  - e.g. **`<!ELEMENT person (name, phone)>`**
- IDREF and IDREFS cannot be constrained, no way to specify the element name of the referenced objects.
- no notion of atomic types, only #PCDATA
  - e.g. cannot specify "Age" is an integer
- no range specification
  - e.g. cannot specify the range of Age is from 0 to 140
- Element and child element names are global.
- can be too vague:
  - e.g. **`<!ELEMENT person ((name|phone|email)*)>`**

Note: The symbol | means alternation, i.e. "or".

XML Namespace

- XML Namespaces provide a method to avoid element name conflicts.

- Name Conflicts
  - In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.
  - This XML carries HTML table information:

```
<table>
  <tr>
   <td>Apples</td>
   <td>Bananas</td>
  </tr>
</table>
```

    This XML carries information about a table (a piece of furniture):

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

  - If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.
  - An XML parser will not know how to handle these differences.

XML Namespace (cont.)

- Solving the Name Conflict Using a Prefix
  - Name conflicts in XML can easily be avoided using a name prefix.
  - This XML carries information about an HTML table, and a piece of furniture:

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

In the example above, there will be no conflict because the two <table> elements have different names.

XML Namespaces

Objectives:

- To differentiate two different elements that happen to share the same name.
- To group elements relating to a common idea together.
- Element name conflicts can be resolved by using different namespaces.
- Just as two different Java classes can have the same name as long as they are defined in two separate packages, two different XML elements can have the same name as long as they belong to two different namespaces.

E.g. A mathematician and a doctor use the term “operation” to mean very different things.

What is namespace?

- The XML namespace specification defines a way to group elements and attribute names so that schemas created by one organization will not conflict with those created by another.
- Each namespace defined in an XML document must be associated with a distinct uniform resource identifier (URI), e.g. http://me.com/namespaces/foobar
- These URIs have no semantic meaning and do not refer to actual web resources.
- Two URIs are considered distinct if they are distinct character strings. Internet URLs are unique so the likelihood of conflicting prefixes is automatically reduced to nil.

What is namespace? (cont.)

- A namespace declaration looks like this:

```
<parent  xmlns:a="http://url1"
         xmlns:b="http://url2"> ... </parent>
```

The namespace http://url1 is bound to the prefix **“a”** and the namespace http://url2 is bound to the prefix “**b**”.

One can use one default namespace and multiple additional namespaces in an XML document and its elements.

So, if you want to write an XML document, and you want to make sure that no one else conflicts with your terminology, use namespace prefixes and associate the prefixes with your very own domain URLs.

What is namespace? (cont.)

- Namespaces have a scope associated with them.
- A namespace declared in a parent element is bound to a given prefix for that element as well as for all of its child elements, unless that prefix is overridden in a child element by being assigned to a different namespace.
- The association between the namespace and prefix declared in an element do not apply to the siblings of that element.

What is namespace? (cont.)

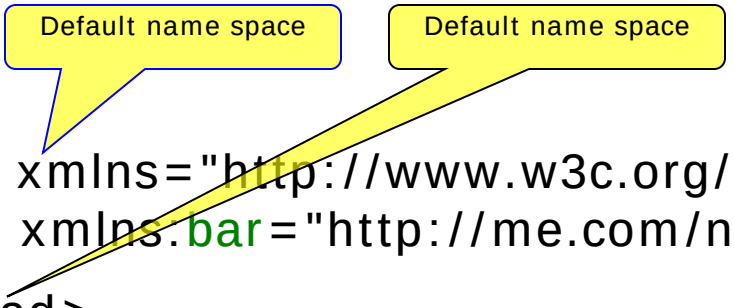
- A default namespace can be defined by omitting the prefix mapping in the declaration.

  e.g. `xmlns="http://url3"`

- Note: Attribute names never inherit the default namespace and must be explicitly mapped to a namespace.

Defining a default name space

E.g.




```
<html  xmlns="http://www.w3c.org/1999/xhtml"
       xmlns:bar="http://me.com/namespaces/foobar" >
   <head>
      <title> An example document </title>
   </head>
   <body>
      <bar:e1> A simple document </bar:e1>
      <bar:e2> Another element </bar:e2>
   </body>
</html>
```

Note: Namespaces provide for scoping using prefixes for the tag names.

The syntax is: <prefix:tagname>

Defining more than one namespace:

```
<foo:tag  xmlns:foo="http://me.com/namespaces/foofoo"
          xmlns:bar="http://me.com/namespaces/foobar" >
    <foo:head>
        <foo:title> An example document  </foo:title>
    </foo:head>

    <bar:body>
        <bar:e1> a simple document </bar:e1>
        <bar:e2> Another element </bar:e2>
    </bar:body>
</foo:tag>
```

Another example for XML Namespace

Note: It is also possible for the same prefix refer to different namespaces *depending on the context*.

```
<a:Envelope
      xmlns="http://default"
      xmlns:a="http://urla"
      xmlns:b="http://urlb"
      xmlns:c="http://urlc" >

   <a:Header xmlns=""
             xmlns:b="http://alturlb">
      <b:type>HelloWorld</b:type>
      <c:to xmlns:c="http://alturlc">Mark Priest</c:to>
      <from>John Smith</from>
   </a:Header>

   <a:Body>
      <text xmlns="http://newdefault">Hello</text>
      <b:mood>Happy</b:mood>
      <c:day>Friday</c:day>
      <month>January</month>
    </a:Body>

</a:Envelope>
```

XML Schema

- Proposed as an enhancement of DTD
- unifies previous schema proposals
- An XML Schema describes the structure of an XML document.
- The XML Schema language is also referred to as XML Schema Definition (XSD).
- two documents: Structures and Datatypes
  - http://www.w3.org/XML/Schema
- XML Schema is very complex
  - Often criticized
  - Some alternative proposals

XML Schema (cont.)

- uses XML syntax and XML namespace
- can define simple type for element and attribute
- has many predefined data types such as string, decimal, integer, boolean, date, time, etc. ID, IDREF, and IDREFS, etc. are derived from the String data type.
- Element and attribute can have default value, fixed value, or required.
- Can specify the range or enumeration constraints for element and attribute value.

XML Schema (cont.)

- Element can be simple element or complex element which contains other elements and/or attributes
- complex elements may contain both other elements and text (mixed="true").
- Can define minimum and maximum occurrences of a child element
- Can define child elements of an element to be ordered (sequence) or unordered (all) or choice.

XML Schema (example)

Namespace of XML schema

Default value for both minOccurs & maxOccurs are 1

Choose one

```
<xs:element name="paper">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title"  type="xs:string" />
      <xs:element name="author" type="xs:string"
                  minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="year" type="xs:integer"/>
      <xs:choice> <xs:element name="journal" type="xs:string"/>
                  <xs:element name="conference" type="xs:string"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element
```

DTD:

```
<!ELEMENT paper(title,author*,year,(journal|conference))>
```

XML Schema (example cont…)

Note: Alternatively, element paper can be defined as complex type papertype.

```
<xs:element name="paper" type="papertype"/>

<xs:complexType name="papertype">
   <xs:sequence>
      <xs:element name="title"  type="xs:string"/>
      <xs:element name="author" type="xs:string"
                  minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="year" type="xs:inetger"/> />
      <xs:choice> <xs:element name="journal" type="xs:string"/>
                  <xs:element name="conference" type="xs:string"/>
      </xs:choice>
   </xs:sequence>
</xs:complexType>
```

DTD:

```
<!ELEMENT paper(title,author*,year,(journal|conference))>
```

Elements *vs.* Types in XML Schema

```
<xs:element name="person">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="name"
               type="xs:string"/>
   <xs:element name="address"
               type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

The type of element person is defined inside the element person.

```
<xs:element name="person"
            type="ttt"/>

<xs:complexType name="ttt">
 <xs:sequence>
  <xs:element name="name"
              type="xs:string"/>
  <xs:element name="address"
              type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

The type of element person is ttt, and the type ttt is defined somewhere else.

DTD: **<!ELEMENT person (name, address)>**

Elements *vs.* Types in XML Schema (cont…)

- Types:
  - Simple types (integers, strings, …)
  - Complex types (regular expressions, like in DTDs)
- Element-type-element alternation:
  - Root element has a complex type
  - That type is a regular expression of elements
  - Those elements have their complex types
  - …
  - On the leaves we have simple types

Local and Global Types in XML Schema

- Local type:

```
<xs:element name="person">
        [define locally the person's type]
</xs:element>
```

- Global type:

```
<xs:element name="person"  type="ttt"/>

<xs:complexType name="ttt">
        [define globally the type ttt]
</xs:complexType>
```

Note: Global types can be reused (shared) in other elements. Global elements are like in DTDs.

Note: Several elements can refer to the same complex type, like this:

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student"  type="personinfo"/>
<xs:element name="member"   type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Note: We can also base a complex element on an existing complex element and add some elements, like this:

```
<xs:element name="employee" type="fullpersoninfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city"    type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Regular Expressions in XML Schema

- Recall the element-type-element alternation:

```
<xs:complexType name="…">
        [regular expression on elements]
</xs:complexType>
```

- Regular expressions:
  - `<xs:sequence> A B C </…>` = `A B C`
  - `<xs:choice> A B C </…>` = `A|B|C`
  - `<xs:all> A B C </…>` (child elements are not ordered)
  - `<xs:… minOccurs="0" maxOccurs="unbounded"/>..</…>` = `(…)*`
  - `<xs:… minOccurs="0" maxOccurs="1"/>..</…>` = `(…)?`

Summary of XML Schema

- Formal Expressive Power:
  - can express precisely the regular tree languages (over unranked trees)
- Lots of other stuff
  - some form of inheritance
  - a “null” value
  - unique value
  - large collection of data types
- **Note:** XML schema can only express 1:1 and 1:n binary relationships but not m:m. It also cannot express n-ary relationships where n>2. In fact, XML Schema and DTD only describe the hierarchical structure of the XML data. No concepts of relationship type and relationship attribute.

  See ORA-SS lecture notes for more details.

Summary of XML Schema (cont…)

- In semistructured data:
  - graph theoretic (because of IDREF(S)
  - data and schema are decoupled
  - used in data processing
- In XML
  - from grammar to object-oriented
  - schema wired with the data
  - emphasis on semantics for exchange