158258 Web Development
Constraint Models for XML Documents — XML Schema Definition Language (XSDL)

Computer Science and Information Technology
School of Mathematical & Computational Sciences
Massey University
(AKLI, DISD & MTUI)

Revised: 2022-08-03

Videos for this Topic

1. This slide deck: Video
2. Live session videos:
    - *Lecture*
    - *Tutorial*
    - *Lab Practical*

Aim and Objectives

- The aim of this topic is to present a method for defining a custom mark-up language in XML by using the XML Schema Definition Language (XSDL) as one instance of a constraint model for XML vocabularies.

- At the end of this topic, your should be able to:

    1. *Discuss the pros and cons of DTDs;*
    2. *Describe the syntax of XML Schema Definition Language (XSDL);*
    3. *Design a schema for your mark-up language;*
    4. *Write a schema for your mark-up language in XSDL; and*
    5. *Validate an XML document against the rules specified in the XML Schema.*

Document Type Definitions (DTD)

The DTD entities defined in file, `codes.dtd`:

```
<!ENTITY BF100P "Butterfly farm pop-up self-
      erecting portable greenhouse">
<!ENTITY BFGK10 "Field of Dreams backyard
      butterfly garden kit">
<!ENTITY HME100 "Hummingbird Hawkmoth (Manduca
      Sexta), 100 eggs">
<!ENTITY MBL25 "Monarch Butterfly, 6-12 larvae">
<!ENTITY MP12 "Monarch Pupae (Danaus Plexippus),
      12 pupae">
<!ENTITY MWT15 "Giant Milkweed Tree (Calotropis
      Ssp.), 1 crown flower">
<!ENTITY PLBK70 "Painted Lady classroom breeding
      kit, 70 larvae">
```

DTD Schema, `customers.dtd`, for XML Customer Documents:

```
```

Example XML document:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"
        ?>

<customers>
   <customer custID="cust201" custType="home">

      <name title="Mr.">David Lynn</name>

      <address>
         <![CDATA[
         211 College Street
         Awapuni, Palmerston North 4412
         ]]>
      </address>

      <phone>(06) 555-1812</phone>
      <email>dlynn@nhx.nz</email>

      <orders>

         <order orderID="or1031" orderBy="cust201">
            <orderDate>8/1/2012</orderDate>
```

```
<!ELEMENT customers (customer+)>
<!ELEMENT customer (name, address, phone, email?,
      orders)>
<!ATTLIST customer custID ID #REQUIRED>
<!ATTLIST customer custType (school | home |
      business) #IMPLIED>

<!ELEMENT name (#PCDATA)>
<!ATTLIST name title (Mr. | Mrs. | Ms.) #IMPLIED>

<!ELEMENT address (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT orders (order+)>

<!ELEMENT order (orderDate, items)>
<!ATTLIST order orderID ID #REQUIRED>
<!ATTLIST order orderBy IDREF #REQUIRED>

<!ELEMENT orderDate (#PCDATA)>
<!ELEMENT items (item+)>

<!ELEMENT item (#PCDATA)>
<!ATTLIST item itemPrice CDATA #REQUIRED>
<!ATTLIST item itemQty CDATA "1">

<!-- codes.dtd contains a list of product codes (IE
      browser only) -->
<!ENTITY % itemCodes SYSTEM "codes.dtd">
%itemCodes;
```

```
            <items>
               <item itemPrice="299.95">&BF100P;
         </item>
               <item itemPrice="49.95">&BFGK10;
         </item>
            </items>
         </order>

         <order orderID="or1142" orderBy="cust201">
            <orderDate>9/14/2012</orderDate>
            <items>
               <item itemPrice="52.23" itemQty="2">&
         MWT15;</item>
               <item itemPrice="124.44"
         itemQty="3">&MBL25;</item>
            </items>
         </order>

      </orders>
   </customer>
...
</customers>
```

DTD Entities: Examples

In DTDs, you could define the Entity:

```
<!ENTITY wow "Wonders of the World">
```

You could then use the entity in your XML document for commonly used text:

```
<story>
The first and most interesting fact about the
         gardens is that there is significant
controversy about whether the gardens existed at
         all.
...
Regardless of the final outcome, it is interesting
         to note that the imagination
of the poets and ancient historians have created one
         of the &wow;.
</story>
```

You could also define an entity for text is in a file, `gardens.net`, as follows:

```
<!ENTITY gardens_story SYSTEM "gardens.ent">
```

You could then use the entity in an XML document as follows:

```
<wonder>
  <name language="English">Hanging Gardens of
         Babylon</name>
  <location>Al Hillah, Iraq</location>
  <height units="feet">0</height>
  <history>
    <year_built era="BC">600</year_built>
    <year_destroyed era="BC">226</year_destroyed>
    <how_destroyed>earthquake</how_destroyed>
    &gardens_story;
  </history>
```

```
  ...
</wonder>
```

Pros and Cons of DTDs

DTD Advantages

1. They are compact and easily comprehended with a little direction;
2. They can be defined inline (internal DTD, for quick development);
3. They can define entities;
4. They are likely the most widely accepted and commonly supported(?)

DTD Disadvantages

1. Not written using XML syntax, and
2. Require parsers to support an additional language;
3. No support for Namespaces;
4. No data typing, thereby decreasing the strength of the validation;

by most XML parsers

5. They have limited capacity to define how many child elements can nest within a given parent element.

XML Schema Basics: *Introduction* — *History & Characteristics*

History

- 2001, W3C developed XML Schema to address DTD limitations (NB: DTDs are also XML schema)
- -XML Schema a.k.a XML Schema Definition (XSD)
- Current version 1.1 – now called XML Schema Definition Language (XSDL)

Characteristics

- Written in XML;
- Deeper and more powerful than DTDs:
  - *Data types;*
  - *Namespaces;*
  - *Local and global elements;*
- More control over

- Most widely recognised name is still XML Schema.

contents of XML documents

- Expected to replace DTDs as the most popular constraint model for XML

XML Schema Basics: *Working With XML Schema*

XML Schema definition

XML Schema for a World Wonder, `wonder.xsd`:

```
<?xml version="1.0"?>
  <!-- XML Schema namespace -->
<xs:schema xmlns:xs="http://www.w3.org
         /2001/XMLSchema">

  <!-- XS prefix demanded by namespace -->

  <xs:element name="wonder">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="string"/>
        <xs:element name="location" type="string"/>
        <xs:element name="height" type="string"/>
      </xs:sequence>
```

XML document

XML document, `wonder.xml`, that uses the language defined in ”’wonders.xsd```:

```
<?xml version="1.0"?>

  <!--  Allows to specify location of XML Schema -->

<wonder xmlns:xsi="http://www.w3.org/2001/XMLSchema-
         instance"
  xsi:noNamespaceSchemaLocation="wonder.xsd">

    <!--Location of XML Schema file, above -->

  <name>Colossus of Rhodes</name>
  <location>Greece</location>
  <height>107</height>
```

```
    </xs:complexType>
  </xs:element>

  <!-- Root element for all XML Schema definitions
        -->

/xs:schema>
```

```

</wonder>
```

Defining Simple Types

Basics

- Simple type element contain a value and can't have children;
- -XML Schema has large collection of built-in simple types: strings, boolean, URLs, date, time numbers, etc
- Restrictions to simple types are called facets – limit simple types, e.g., strings limited to e-mails only

Example

```
<xs:element name="height" type="xs:string"/>
<xs:element name="year_built" type="xs:integer"/>
```

Simple Types

- `xs:string` — string of chars
- `xs:boolean` — for values true and false
- `xs:decimal` — decimal numbers
- `xs:date` — for date

elements

- `xs:time` — time of day
- `xs:anyURI` — elements that contain reference to file on Internet, LAN and computer

Date and Time Types

- `xs:date` — YYYY-MM-DD
- `xs:time` — Hh:mm:ss
- `xs:dateTime` — yyyy-mm-ddThh:mm:ss,
  - *e.g. 2008-05-23T16:22:00*
- `xs:duration` — PnYnMnDTnHnMnS,
  - *e.g., P3M4DT6H17M – 3months 4 days 6 hours and*

- `xs:gYear` — yyyy
- `xs:gYearMonth` — "yyyy-mm"
- `xs:gMonth` — "–mm"
- `xs:gMonthDay` — "–mm-dd"
- `xs:gDay` — "—dd"

NOTES

- "g" stands for Gregorian calendar

17 minutes.

- All time types can end with optional time zone indicator:
  - *Z for UTC*
  - *hh:mm or +hh:mm for offset from UTC*

Number Types

- xs:decimal
- xs:integer
- xs:positiveInteger
- xs:negativeInteger
- xs:int — signed 32-bit integer
- xs:float — single precision 32-bit floating-point numbers, e.g., 43e-2

XML Schema

```
<xs:element name="years_standing"
         type="xs:positiveInteger"/>
<xs:element name="height" type="xs:decimal"/>
```

XML Document

```
<years_standing>1602</years_standing>
<height>384.25</height>
```

Predefining an Element's Content

Fixed Value

- XML Schema Fragment:

```
<xs:element name="how_destroyed" type="xs:string"
         fixed="fire"/>
```

- Are the following XML fragments correct and why/why not?

1. *<how_destroyed>fire</how_destroyed>*
2. *<how_destroyed>*
   </how_destroyed>
3. *<how_destroyed>earthquake</how_destroyed>*

Default Value

- XML Schema Fragment:

```
<xs:element name="how_destroyed" type="xs:string"
         default="fire"/>
```

- Are the following XML fragments correct and why/why not?

1. *<how_destroyed>fire</how*
2. *<how_destroyed>*
   </how_destroyed>
3. *<how_destroyed>earthquak*

Deriving Custom Simple Types

XML Schema

- Name of new type, "story_type". If absent, we've anonymous type.

```
<xs:simpleType name="story_type">
<xs:restriction base="xs:string">
<xs:length value="1024"/>
</xs:restriction>
</xs:simpleType>
```

- The custom type defined above can be re-used for any other element in the XML schema.

XML Document

```
<xs:element name="story" type="story_type" />
<xs:element name="summary" type="story_type" />
<xs:element name="another_story" type="story_type"
        />
```

- The new `story_type` custom type can now be used in as many element definitions as you would like
- Note that you refer to the custom type as story_type and not as `xs:story-type`.

- Notice how the `xs:simpleType` element's name attribute is set to `story_type`.
- `story_type` is the name that can be used to reference the newly defined custom type!
- Anonymous types can only be used inside the element in which its defined

- `xs:` prefix refer to the XML Schema namespace

Specifying a Range of Acceptable Values

XML Schema

```
<xs:element name="game_day">
  <xs:simpleType>
    <xs:restriction base="xs:date">
      <xs:minInclusive value="1954-04-13" />
      <xs:maxInclusive value="1976-10-03" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML document

- Are the following valid/invalid?

1. *<game_day>1976-07-20</game_da*
2. *<game_day>2008-07-04</game_da*

Specifying a Set of Acceptable Values

XML Schema

```
<xs:element name="wonder_name">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Colossus of
         Rhodes" />
      <xs:enumeration value="Great Pyramid of
         Giza" />
      <xs:enumeration value="Hanging Gardens of
         Babylon" />
      <xs:enumeration value="Statue of Zeus at
         Olympia" />
      <xs:enumeration value="Temple of Artemis
         at Ephesus" />
      <xs:enumeration value="Mausoleum at
         Halicarnassus" />
      <xs:enumeration value="Lighthouse of
         Alexandria" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- Each enumeration value

XML Document

Are the following valid/invalid? Why?

1. `<wonder_name>Great Pyramid of Giza</wonder_name>`
2. `<wonder_name>Great Pyramid</wonder_name>`
3. `<wonder_name>Lighthouse of Alexandria, Hanging Gardens of`

must be unique

- Enumeration values may contain white space
- You can use `xs:enumeration` facet with all simple type except boolean

```
Babylon</wonder_name>
```

Limiting Length of an Element

XML Schema

- **length** facet is used with string-based simple types:

```
<xs:element name="wonder_code">
  <xs:simpleType>
    <xs:restriction base="xs:string">
    <xs:length value="5" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- **minLength** also available. Use non-negative integer values:

XML Document

- Example of he womder_code element:

```
<?xml version="1.0"?>
<simple_types xmlns:xsi="http://www.w3.org
        /2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="10-37.xsd">

  <wonder_code>w_285</wonder_code>

</simple_types>
```

- Example of the brief_description element:

```
<xs:element name="brief_description">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="256" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<brief_description>
In 294 BC, a huge statue was built honouring the god
        Helios.
This Colossus of Rhodes, often depicted straddling
        the harbour, likely stood by it.
The statue was toppled by earthquake, and was not
        rebuilt.
Even broken, many still travelled to see it.
</brief_description>
```

Specifying a Pattern for an Element

Regular expressions

- `.` (a period) — any character;
- `\d` — any digit;
- `\D` — any non-digit;
- `\s` — any white space;
- `\S` — character not a white space;
- `X*` — zero or more x's
- `X?` — one or more x's

XML Schema

The Regular Expression is declared here:

```
<xs:element name="wonder_code">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="w_\d{3}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- What does this regular expression mean here?

- `X+` — one or more x's
- `[abc]` — one of a group of values a, b, or c
- `[0-9]` — range of values from 0 to 9
- `this | that` — this or that included
- `X{5}` — exactly 5 x's
- `X{5,}` — at least 5 x's
- `X{5,8}` — at leat 5 x's and at most 8 x's
- `(xyz){2}` — exactly 2 xyz's in a row

Basics of Complex Types

Four Complex Types

1. **Text only** — complex type element with complex content, children and attributes;
2. **Element only** — element type element with complex content, children and attributes
3. **Empty element** — complex type element

Definition and Rationale

- Complex Type contain:
  - *child elements, attributes, or a combination of the two;*
- There is debate about complexity of these types;
- Reasons for using complex types in XML:

with complex content — contains attributes;

4. **Mixed content** — complex type element with both complex content and simple content

1. Allow root element to have children of its own;

2. Allow elements to have attributes

Elements vs Complex Types

- Elements and complex types both define *sub-trees*.
- Elements are *standalone* — they can be used as root elements.
- On the other hand, complex types can only occur within elements.
- This allows to define the root.

Deriving Complex Type

Definition of `year-built` element with an attribute, a complex type derived from extension of simple type with an attribute:

```
<xs:element name="year_built">
<xs:complexType>
<xs:simpleContent>
          <xs:extension base="xs:positiveInteger">
    <xs:attribute name="era" type="xs:string"/>
         </xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
```

- Example XML document:

```
<year_built era="BC">
```

Definition of the `ancient_wonders` element, a complex type derived from `complexContent` that restricts anyType:

```
<xs:element name="ancient_wonders">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:element name="wonder"
         type="wonderType" />
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

- Example XML document:

```
    282
</year_built>

<year_destroyed era="BC">
    226
</year_destroyed>
```

```
<ancient_wonders>
    <wonder>
        ...
    </wonder>
</ancient_wonders>
```

Structure of Complex Types

- Within a complex types, the following groupings are permitted:
  - `xs:all` — *children can appear zero or one times in any order*
  - `xs:sequence` — *children can appear one or many times, and the order is enforced*
  - `xs:choice` — *only one children can appear, but it can appear multiple times.*
- Cardinalities can further be enforced using `minOccurs/maxOccurs` attributes.
- Already defined elements can be referenced using the ref attribute in `xs:element`.

XML Namespaces

- XML namespaces are used to scope elements.
- This is useful to "mix" elements from different vocabularies.
- Scoped elements are written with a namespace prefix: `prefix:name`. This is called a qualified name or `qname`.
- Since ":" can be used in XML names, none namespace aware applications (older

- While the prefix is arbitrary *(but the full name must be a valid XML name)*, there are de-facto standards for common name spaces such as: `xs/xsd (XMLSchema)`, `xsl` (XML Transformations), `dc`, `rdf`, etc.
  - *A default name space can be defined as well by using the `xmlns` attribute*

parsers) are compatible with namespaces.

- The namespace prefix is declared using the `xmlns:<prefix>` attribute in any element which is an ancestor of the elements using the prefix.
- To make the prefix unique, it is mapped to a URI
- See also http://www.w3.org/TR/1999/REC-xml-names-19990114

(without a prefix!) in the root element:

- *All elements will then automatically get this name space(e.g., when a name space aware parser reads the document).*

XML Schema Namespace

- The name space definition applies to the element and (recursively) to its children.

```
<xs:schema xmlns:xs="http://www.w3c.org/2001/XMLSchema"

    <xs:element name="attachment">
```

Topic Summary

- In this topic, we have covered the following:

    1. *Document Type Definitions (DTD)*
        - Pros and Cons of DTDs
    2. *XML Schema:*
        - Defining Simple Types
        - Defining Complex Types
    3. *Namespaces*

Tutorial — XML Schema

- Task: **Write an XML Schema Definition for E-mail Messages.** Use the design presented in the following diagram.