XML Schema Definition Language (XSDL)

Constraint Models for XML Documents:

Kuda Dube <k.dube@massey.ac.nz>
Computer Science and Information Technology
School of Engineering & Advanced Technology
Massey University

Topic Outline

- Aim
- Learning Goals
- Pros and Cons of DTDs
- XML Schema:
  - Defining Simple Types
  - Defining Complex Types
  - Namespaces
  - Case Study
- End

Topic Aim

- To present a method for defining a custom mark-up language in XML by using the XML Schema Definition Language as one instance of a constraint model for XML vocabularies.

Learning Objectives

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.

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(?)* by most XML parsers

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

Pros and Cons of DTDs: Entities

DTD

XML

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

```
<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>
```

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

```
<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>
…
```

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)
- Most widely recognised name is still XML Schema.

CHARACTERISTICS

- Written in XML;
- Deeper and more powerful than DTDs:
  - Data types;
  - Namespaces;
  - Local and global elements;
- More control over 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 version="1.0"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/X
MLSchema">
<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>
    </xs:complexType>
</xs:element>
/xs:schema>
```

XML Schema namespace

XS prefix demanded by namespace

Root element for all XML Schema definitions

XML DOCUMENT

```
<?xml version="1.0"?>
<wonder
    xmlns:xsi="http://www.w3.
    org/2001/XMLSchema-
    instance"
    xsi:noNamespaceSchemaL
    ocation="09-06.xsd" >
  <name>Colossus of
    Rhodes</name>
  <location>Greece</location>
  <height>107</height>
</wonder>
```

Allows to specify location of XML Schema

Location of XML Schema file

Defining Simple Types: Example **

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"/>
```

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-dd**T**hh:mm:ss
  - E.g. 2008-05-23T16:22:00
- **xs:duration**
  - **PnYnMnDTnHnMnS**
  - E.g., P3M4DT6H17M – 3months 4 days 6 hours and 17 minutes.

- Xs:gYear – yyyy
- Xs:gYearMonth – "yyyy-mm"
- Xs:gMonth – "--mm"
- Xs:gMonthDay – "--mm-dd"
- Xs:gDay – "---dd"

NOTES

- ❑ "g" stands for Gregorian calendar
- ❑ All time types can end with optional time zone indicator:
  - Z for UTC
  - -hh:mm or +hh:mm for offset from UTC

Number Types

1. **xs:decimal**
2. **xs:integer**
3. **xs:positiveInteger**
4. **xs:negativeInteger**
5. **xs:int** – signed 32-bit integer
6. **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_destroyed>
2. <how_destroyed></how_destroyed>
3. <how_destroyed>**earthquake</**how_destroyed>

Deriving Custom Simple Types

XML SCHEMA

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

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

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

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**.
- ❑ "**xs:**" prefix refer to the XML Schema namespace

Specifying a Range of Acceptable Values

XML SCHEMA	XML DOCUMENT
<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>	▪ Are the following valid/invalid? 1. `<game_day>1976-07-20</game_day>` 2. `<game_day>2008-07-04</game_day>`

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 must be unique
- ❑ Enumeration values may contain white space
- ❑ You can use **xs:enumeration** *facet* with all simple type except **boolean**

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 Babylon** `</wonder_name>`

Limiting Length of an Element

XML SCHEMA

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

length facet is used with string-based simple types.

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

minLength also available. Use non-negative integer values

XML DOCUMENT

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

<!-- to follow this example, see the section below -->
     <wonder_code>w_285</wonder_code>

</simple_types>
```

```
<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 wasn't 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
- **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

XML SCHEMA

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

This is a ***Regular Expression.***
What does it mean here?

Basics of Complex Types

DEFINITION & 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:
  1. *Allow root element to have children of its own;*
  2. *Allow elements to have attributes*

FOUR COMPLEX TYPES

1. **Text only** – complex type element with complex content, children & attributes;
2. **Element only** – element type element with complex content, children & attributes
3. **Empty element** – complex type element with complex content – contains attributes;
4. **Mixed content** – complex type element with both complex content and simple content

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

XML SCHEMA

XML DOCUMENT

```
<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>
```

Complex type derived from extension of simple type with an attribute.

```
<year_built era="BC">
       282
</year_built>

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

```
<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>
```

Complex type derived from **complexContent** that restricts **anyType**.

```
<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 - 1

- ❑ **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 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

XML Namespaces - 2

- 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 (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 Name Spaces – 3: *A Hack for DTDs*

- If only one fixed name space is used, it can be declared in the DTD as fixed xmlns attribute of the root element.

```
<!ATTLIST element xmlns CDATA #FIXED “aURI”>
```

- DTDs and name spaces are completely independent.
- DTDs can declare prefixed names.
- Parameter entity references can be used to improve the maintainability of DTSs using name spaces.

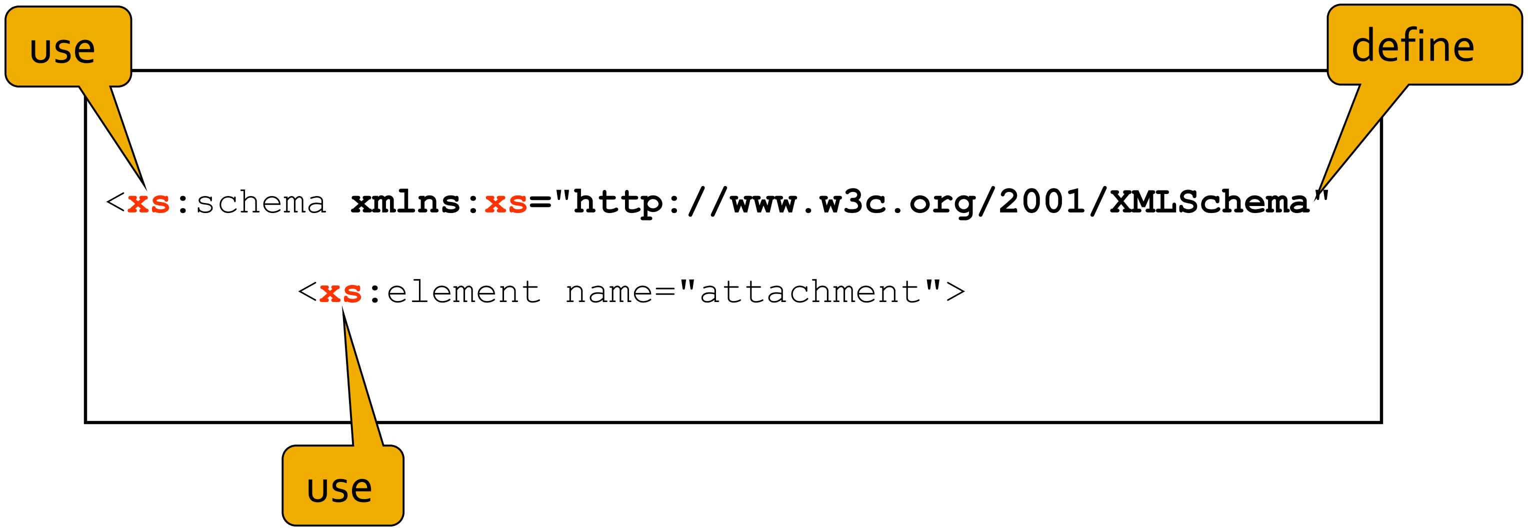
```
<!ENTITY % mail “mail:”>
…
<!ENTITY % mail-to “%mail;to”>
<!ENTITY % mail-cc “%mail;to”>
…
<!ELEMENT %mail-to; (#PCDATA)>
```

Param entity for prefix

Param entities for qualified names

Define elements, can reference param entities

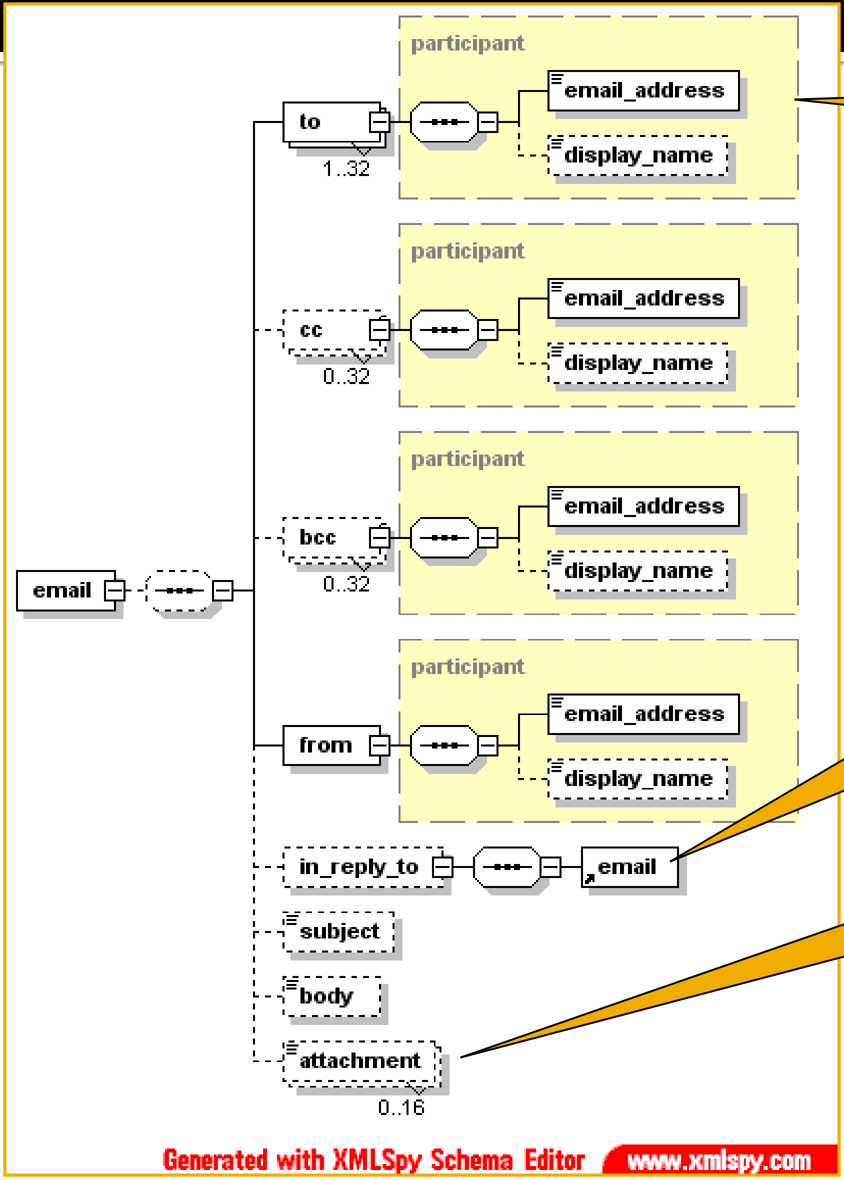
XML Namespaces – 4:
The XML Schema Namespace



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

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

Exercise 3: *Write an XML Schema Definition for E-mail Messages*



Complex type uses

XML Spy View Diagram for E-Mail

(self) reference

Shapes indicate cardinality

End of Lecture & Recap

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