

XML Validation

Ramesh Sankaranarayanan

Department of Computer Science
Australian National University

COMP3410 IT in E-Commerce

Outline

- 1 Introduction
 - Specifying the data model
- 2 Document Type Definition
 - DTD Syntax
 - Referencing DTD's
 - External DTD's
 - Internal DTD's
 - Disadvantages
- 3 XML Schema
 - Introduction
 - XML Schema Syntax
 - Namespaces
 - Referencing Schemas

Outline

- 1 Introduction
 - Specifying the data model
- 2 Document Type Definition
 - DTD Syntax
 - Referencing DTD's
 - External DTD's
 - Internal DTD's
 - Disadvantages
- 3 XML Schema
 - Introduction
 - XML Schema Syntax
 - Namespaces
 - Referencing Schemas

Why specify the data model?

Consider the following example:

```
<catalogue>
  <book>
    <title>XML Unleashed</title>
    <author>Michael Morrison, et al.</author>
    <isbn>0-672-31514-9</isbn>
    <publisher>Sams Publishing</publisher>
    <year>1999</year>
  </book>
</catalogue>
```

Why specify the data model?

This has:

This has a main element **catalogue**, which in turn consists of one (possibly more) **book** elements. Each book element has five sub-elements **title**, **author**, **isbn**, **publisher** and **year**.

Now, look at the following:

```
<book>
  <title></title>
  <author>Michael Morrison, et al.</author>
  <isbn>0-672-31514-9</isbn>
  <publisher>Sams Publishing</publisher>
  <year>1999</year>
</book>
```

Why specify the data model?

In this case

- It is a well-formed document
- But, a required element, **title**, is empty
- The only way an application can figure this out is if it is provided with a specification of the underlying data model

Specifying the data model

Two commonly used techniques are:

- Document Type Definition (DTD)
- XML Schema

Outline

- 1 Introduction
 - Specifying the data model
- 2 Document Type Definition
 - DTD Syntax
 - Referencing DTD's
 - External DTD's
 - Internal DTD's
 - Disadvantages
- 3 XML Schema
 - Introduction
 - XML Schema Syntax
 - Namespaces
 - Referencing Schemas

DTD Example

A DTD for the book document structure

```
<!ELEMENT catalogue (book+)>
<!ELEMENT book (title, author, isbn, publisher,
year?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT year (#PCDATA)>
```

DTD Syntax

The DTD

- Is inherited from SGML
- Uses **EBNF (Extended Backus-Naur Form)** to specify the rules

DTD Structure

Every DTD must have a **root** element, which in this case is **catalogue**. The line

```
<!ELEMENT catalogue (book+)>
```

says that the root element **catalogue** consists of one or more **book** elements.

DTD Syntax

DTD Structure

The line

```
<!ELEMENT book (title, author, isbn,
publisher, year?)>
```

says that the **book** element consists of the stated five sub-elements, which have to appear in the same order, of which **year** may or may not occur. The rest have to appear exactly once.

DTD Syntax

DTD Structure

The line

```
<!ELEMENT title (#PCDATA)>
```

indicates that the element **title** consists of **parsed character data**.

DTD Syntax

Adding attributes

```
<!ATTLIST book
  type (adventure | mystery | romance | sci-fi)
  "mystery"
  rating (G | PG | M | X) "G"
  review (1 | 2 | 3 | 4 | 5) "3">
```

adds attributes **type**, **rating** and **review** to the **book** element. Each of these is an **enumerated** type and has an associated **default** value.

Referencing DTD's

How to specify the DTD in an XML document?

There are two ways of doing this.

- **External** DTD's
- **Internal** DTD's

External DTD's

Assume that the DTD is stored in a file called **catalogue.dtd**. We can then refer to it from within an XML document as follows:

```
<?xml version="1.0"?>
<!DOCTYPE catalogue SYSTEM "catalogue.dtd">
```

The above states that the root element of the XML document is **catalogue** and that the DTD is contained in an external file **catalogue.dtd**.

External DTD's

Some advantages:

- The DTD is reusable.
- The document is cleaner.

Internal DTD's

Contained within the document itself

```
<?xml version="1.0"?>
<!DOCTYPE catalogue [
  <!ELEMENT catalogue (book+)>
  ...
]>
```

You can use both external and internal DTD's in the same document. If there are elements with the same name, then the internal one overrides the external one.

Disadvantages of DTD's

Some of the disadvantages are:

- They are based on a specialised syntax
- They don't support many data types
- They don't support namespace integration
- They have a closed data model
- They only have a primitive form of grouping

Internal DTD's

Internal DTD's are used when:

- Only a single document is being created
- There is a need to minimize overhead

Outline

- 1 Introduction
 - Specifying the data model
- 2 Document Type Definition
 - DTD Syntax
 - Referencing DTD's
 - External DTD's
 - Internal DTD's
 - Disadvantages
- 3 XML Schema
 - Introduction
 - XML Schema Syntax
 - Namespaces
 - Referencing Schemas

Introduction

XML Schema

XML-Data A note issued in January 1998.

DCD or Document Content Description Another note issued in July, 1998.

W3C XML Schema Issued in May 1999 as a working draft. XML Schema 1.0 became a recommendation in May, 2001

W3C XML Schema XML Schema 1.1 is a candidate recommendation as of April 2009.

XML Schema Syntax

XML Schema for catalog example (catalog.xsd)

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

  <!-- Define the root element as catalogue-->
  <element name="catalogue" type="catalogueType"/>
  <!-- Consists of one or more books -->
  <complexType name="catalogueType">
    <sequence>
      <element name="book" type="bookType"
        minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
```

XML Schema Syntax

XML Schema for catalog example...

```
<!-- Define bookType -->
<complexType name="bookType">
  <sequence>
    <element name="title" type="string"/>
    <element name="author" type="string"/>
    <element name="isbn" type="string"/>
    <element name="year" type="gYear"/>
  </sequence>
</complexType>

</schema>
```

XML Schema Syntax

Here

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

<schema xmlns="http://www.w3.org/2001/XMLSchema">
```

schema serves as the root element for all XML Schema documents.
xmlns specifies the **namespace** where W3C's XML Schema implementation is to be found as the **default** namespace.

XML Schema Syntax

Defining an element

```
<!-- Define the root element as catalogue-->
<element name="catalogue" type="catalogueType"/>
```

We define the element **catalogue** whose **type** is **catalogueType**.

XML Schema Syntax

Defining an element

```
<!-- Consists of one or more books -->
<complexType name="catalogueType">
  <sequence>
    <element name="book" type="bookType"
      minOccurs="1" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

Since **catalogue** consists of one or more **book** elements, we define **catalogueType** to be a **complexType**. It consists of one or more **book** elements, each of which is of **bookType**.

XML Schema Syntax

Defining a complexType

```
<!-- Define bookType -->
<complexType name="bookType">
  <sequence>
    <element name="title" type="string"/>
    <element name="author" type="string"/>
    <element name="isbn" type="string"/>
    <element name="year" type="gYear"/>
  </sequence>
</complexType>
```

We define the complexType **bookType** here. It consists of the elements **title**, **author**, **isbn** and **year**. The first three are of type **string**, while the last one is of type **gYear**.

XML Schema Syntax

Adding attributes

```
<!-- Add an attribute called rating to bookType -->
<complexType name="bookType">
  <sequence>
    <element name="title" type="string"/>
    ...
  </sequence>
  <attribute name="rating" type="string"/>
</complexType>
```

Here, we define the attribute **rating** to be of type **string**. We can also define it as an enumeration with a default value, as shown below.

XML Schema Syntax

Attribute with enumeration

```

<!-- Add an attribute called rating, with choice
of values from a list -->
<attribute name="rating" use="optional"
  default="G">
  <simpleType>
    <restriction base="NMTOKEN">
      <enumeration value="G" />
      <enumeration value="PG" />
      <enumeration value="M" />
      <enumeration value="X" />
    </restriction>
  </simpleType>
</attribute>

```

Why namespaces?

Consider the following case:

There are two different schemas for a **catalog** and they both use the element **rating**, but with entirely different values. If XML documents that use these two schemas were put together in a single document, it would create an element name conflict. **Namespaces** provide a means of avoiding such conflicts. As per W3C specification, a namespace must be a **Uniform Resource Identifier (URI)**. A common URI that is used is the well known **Uniform Resource Locator (URL)**.

Using namespaces

An example

```

<?xml version="1.0" ?>
<xsi:schema
  xmlns:xsi="http://www.w3.org/2001/XMLSchema">

  <xsi:element name="catalogue" .../>
  <xsi:complexType name="catalogueType">
    <xsi:sequence>
      <xsi:element name="book" type="bookType" .../>
    </xsi:sequence>
  </xsi:complexType>
  ...
</xsi:schema>

```

Defining your own namespace

An example

```

<?xml version="1.0" ?>
<xsi:schema
  xmlns:xsi="http://www.w3.org/2001/XMLSchema"
  xmlns="file://students/uxxxxxxx/comp3410/lab2"
  targetNamespace
    ="file://students/uxxxxxxx/comp3410/lab2"
  elementFormDefault="qualified">

  <xsi:element name="catalogue"
    type="catalogueType"/>
  <xsi:complexType name="catalogueType">
    ...
  </xsi:schema>

```

Referencing Schemas

No user namespace defined

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

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

Referencing Schemas

With a defined namespace

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

<catalogue
  xmlns="file://students/uxxxxxxx/comp3410/lab2"
  xmlns:xsi
    ="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation
    ="file://students/uxxxxxxx/comp3410/lab2
    catalog.xsd">
  ...
</catalogue>
```