Software Engineering Project

Object-Oriented Programming using Java

XML

Stuart Porter

Email: **sjp@ohm**

- What is XML?
- XML data - the documents
- XML definition, mark I – Document Type Definition (DTD)
- XML definition, mark II - Schema
- XML Parsing in Java

What is XML?

- Easy way to add information to data in a document
- Information added is *metadata*
- Tag each piece of information with the metadata

Have only three types of entity in XML:

- element name
- attribute
- element content

where:

- element name plus attribute(s) form the start tag
- element name forms end tag
- element content is data plus other element(s)

Thus:

```
<element attribute="value">        // start tag
    content
</element>                         // end tag
```

N.B. XML is case sensitive.

Simple Example

- Could write:

  Joe Bloggs is a Third Year MEng student at the University of York who lives on chocolate

- or, alternatively:

  University of York : Student : Joe Bloggs : MEng : Third Year : chocolate

These tell us something about Joe (too much?!)

- BUT, how about:

```
<university name="York">
    <student name="Joe Bloggs">
        <programme>MEng</programme>
        <year>3</year>
        <preference food="chocolate"/>
    </student>
</university>
```

- **university**, **student**, **programme**, **year**, **name**, **preference**, **food** is metadata
- We can easily validate this structure
- Each *element* is naturally an *object*
  - which may encapsulate other *element*'s (*objects*)

XML Data - the Documents

What makes an XML document?

- One element containing everything else
  - *root* element
- All other elements nest within each other
  - no overlapping
- Every *parsed* entity referenced within document is well-formed
- All attribute values within quotes
- Element names must be valid
  - start with letter or underscore
  - may contain letters, numbers, hyphens, periods, underscores

- To use XML 1.0, has to be well-formed according to
  - Extensible Markup Language (XML) 1.0 (Third Ed.)
  - *http://www.w3.org/TR/REC-xml/*

- Root element is significant
  - Allows for more robust transmission over network
  - Easily determine start and end of data

    $\Rightarrow$ from first few characters - probably first packet

Simple Example of XML 1.0

- If we want XML 1.0, can modify our example:

```
<?xml version="1.0"?>
<university name="York">
    <student name="Joe Bloggs">
        <programme>MEng</programme>
        <year>3</year>
        <preference food="chocolate"/>
    </student>
</university>
```

- Now have well-formed XML 1.0 document.

How do we validate this?

XML Definition, Mark I - Document Type Definition (DTD)

- DTD is set of definitions of all elements / attributes allowed in an XML document.
- Allows us to validate the XML
  - possibly: *http://www.stg.brown.edu/service/xmlvalid/*

    ⇒ requires XML document and DTD to be visible from that site.

Two main definitions:

- **`ELEMENT`** to define element type
- **`ATTLIST`** to define attributes assigned to element

- also can have **`ENTITY`** and **`NOTATION`**

`ELEMENT` Definitions

- Simple element with no content:

```
<!ELEMENT apple EMPTY>
```

  - would give:

```
<apple/>
```

  - can still have attributes

- Will generally have content - other elements or data:

```
<!ELEMENT basket (apple, orange)>
```

  - allows us to ALWAYS put 1 `apple` and 1 `orange` in a basket IN THAT ORDER, giving:

```
<basket>
    <apple/>
    <orange/>
</basket>
```

  - a little strange, but good for:

```
<student>
    <firstname>Joe</firstname>
    <lastname>Bloggs</lastname>
</student>
```

How do we mix things up a bit?

Consider:

```
<!ELEMENT basket (owner, (apples|oranges)?, veg*)>
```

giving rise to:

```
<basket>
    <owner>Joe Bloggs</owner>
    <apples>4</apples>
    <veg name="Carrots">4</veg>
    <veg name="Leeks">2</veg>
</basket>
```

- Two sets of operators:

Order Operators	Meaning
,	(comma) strict sequence
|	(pipe) choice

  - determine the order of elements

Cardinality Operators	Meaning
?	Optional - may or may not appear
*	Zero or more
+	One or more

  - determine the presence and number of elements

But, should also define the other elements:

```
<!ELEMENT basket (owner, (apples|oranges)?, veg*)>
<!ELEMENT owner #PCDATA>
<!ELEMENT apples #PCDATA>
<!ELEMENT veg #PCDATA>
```

- where `#PCDATA` defines content to be *parsed character data*

- `#PCDATA` may contain data or other elements

Now, most of:

```
<basket>
    <owner>Joe Bloggs</owner>
    <apples>4</apples>
    <veg name="Carrots">4</veg>
    <veg name="Leeks">2</veg>
</basket>
```

is defined, except for attributes.

`ATTLIST` Definitions

- Simple definition for our `veg` element:

```
<!ATTLIST veg name CDATA #REQUIRED>
```

  - `CDATA` is *character data* (not-parsed)

- Could define `veg` alternatively:

```
<!ATTLIST veg name CDATA #REQUIRED
        quantity CDATA #IMPLIED>
```

Attribute Defaults	Meaning
`#REQUIRED`	Must appear
`#IMPLIED`	May appear
`#FIXED` plus default value	Must always have default value - if attribute does not appear, value is assumed
Default value only	Default value if attribute not shown, other value can be taken if shown.

- So, the `quantity` above might be set to a default value if omitted
  - default value may be obtained in the document or set by the application.

Element or Attribute?

Some guidelines:

- If the data is likely to be displayed (textual/numeric in nature)
  - place it in the body
- If the data is contextual (e.g., font size)
  - set it as an attribute

So, our example:

```
<basket>
    <owner>Joe Bloggs</owner>
    <apples>4</apples>
    <veg name="Carrots">4</veg>
    <veg name="Leeks">2</veg>
</basket>
```

might better be written:

```
<basket>
    <owner>Joe Bloggs</owner>
    <apples>4</apples>
    <veg>
        <name>Carrots</name>
        <quantity>4</quantity>
    </veg>
    <veg>
        <name>Leeks</name>
        <quantity>2</quantity>
    </veg>
</basket>
```

Types of Attribute

We can choose the type of the attribute (partial list):

Attribute Type	Meaning
`CDATA`	Character data
`ID`	Unique name within document
[Enumerated value]	List of values the attribute can take.

- `CDATA` will generally be specified and can contain anything except markup.
- `ID` must be a valid name in XML
- Enumerated values useful where only limited values can be used, e.g.,

```
<!ATTLIST veg name (Carrots | Leeks) #REQUIRED>
```

  - would indicate a choice only between `Carrots` and `Leeks`
  - all other data causes invalid document

Now we know how to validate an XML document.

  - Let us re-consider the student document.

Can incorporate DTD within XML document:

```
<?xml version="1.0"?>
<!DOCTYPE university [
<!ELEMENT university (student+)>
<!ATTLIST university name CDATA #REQUIRED>

<!ELEMENT student (programme, year, preference*)>
<!ATTLIST student name CDATA #REQUIRED>

<!ELEMENT programme (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT preference EMPTY>
<!ATTLIST preference food CDATA #IMPLIED>
]>
<university name="York">
    <student name="Joe Bloggs">
        <programme>MEng</programme>
        <year>3</year>
        <preference food="chocolate"/>
    </student>
</university>
```

- all stored in *uni.xml*

However, this is inappropriate for multiple XML documents with same DTD.

Generally, separate file for DTD:

```
<!ELEMENT university (student+)>
<!ATTLIST university name CDATA #REQUIRED>

<!ELEMENT student (programme, year, preference*)>
<!ATTLIST student name CDATA #REQUIRED>

<!ELEMENT programme (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT preference EMPTY>
<!ATTLIST preference food CDATA #IMPLIED>
```

- in *uni.dtd* at *http://serv/uni.dtd*

Can now use this to validate our XML at any location:

```
<?xml version="1.0"?>
<!DOCTYPE university SYSTEM "http://serv/uni.dtd">
<university name="York">
    <student name="Joe Bloggs">
        <programme>MEng</programme>
        <year>3</year>
        <preference food="chocolate"/>
    </student>
</university>
```

- in *uni.xml*

Problems with DTD's

- Allow for some validation, but
  - e.g., `#CDATA` does not distinguish between, say, *string* or *int* - both are valid.
- Yet, despite this, the syntax of the DTD is not trivial
  - particularly compared with basic XML.

$\Rightarrow$ can we use the XML structure to describe the definition

- self-defining.

XML Definition, Mark II - Schema

Consider the following possible definition for our student:

```
<?xml version="1.0"?>
<schema version="1.0">
  <element name="university">
    <attribute name="name" type="string"/>
    <element name="student">
      <attribute name="name" type="string"/>
      <element name="programme" type="string"/>
      <element name="year" type="int"/>
      <element name="preference">
        <attribute name="food" type="string"/>
      </element>
    </element>
  </element>
</schema>
```

- Well-formed XML with specific set of elements
  - **`schema`**, **`element`**, **`attribute`**
- Now have specification of type of, say, **`year`**
  - now defined as **`int`**
  - allows greater checking
- Simple, but not trivial to read even with indentation
  - continuous series of **`element`** / **`attribute`** elements

W3C XML Schema (early version):

```
<?xml version="1.0"?>
<schema version="1.0">
  <element name="university">
    <annotation>
      <info>
        The particular university.
      </info>
    </annotation>
    <type>
      <attribute name="name" type="string"/>
      <element name="student">
        <annotation>
          <info>
            Particular student.
          </info>
        </annotation>
        <type>
          <attribute name="name" type="string"/>
          <element name="programme"
type="string"/>
          <element name="year" type="int"/>
          <element name="preference">
            <type>
              <attribute name="food"
type="string"/>
            </type>
          </element>
        </type>
      </element>
    </type>
  </element>
</schema>
```

- Now added **`annotation`** and **`type`**
  - **`info`** sub-element of **`annotation`**
- Documentation for particular element now easily included in **`annotation`**
- Elements and attributes now defined within **`type`** for particular **`element`**

- More recent version more specific, more complex
  - could use it if preferred
  - W3C recommendation since 2 May 2001

So, now have:

- distinction between data types, e.g., **`string`**, **`int`**
  - allows for more checking during validation
- Syntax simple
  - if we understand XML syntax, we automatically understand schema's
- Now have self-defining XML documents

Only current drawback is lack of freely available validator's.

Summary

- XML is simple, hierarchical, object-oriented means of storing data
  - tag data with metadata to give meaning
- Conformance to standards gives well-formed document
- DTD gives means of specifying structure of XML file
  - allows for well-formed AND valid document
- Schema's allow for more specific definition of structure
  - more complex specification
- Can always generate our own schema
  - needs to be well-formed XML
- XML well-structured for parsing in Java
  - both have strong object model
  - excellent example of use of `interface`'s