CHAPTER 2

XML PROCESSING

This chapter describes the XML syntax, XML schema description languages, validating XML, query processing on XML etc.

2.1 XML SYNTAX

XML is a technology for creating markup languages to describe any data type virtually in a structured manner. Markup gives the description of storage layout and logical structure of an XML document. The storage units of XML documents are called as entities. Either parsed or unparsed data may be XML entities. The tags used in HTML are predefined. The predefined tags like <p>, <h1> etc. can only be used in HTML documents. But, XML allows users to describe data more precisely by creating their own tags. XML document is usually stored in a text file which has a file name with an extension ‘.xml’. Any text editor or free downloadable software can be used to create and edit XML documents. Figure 2.1 shows the sample XML document “course.xml” and its tree representation. This sample document is taken as a running example throughout this thesis.
Figure 2.1 A sample XML document and its tree representation

The XML document must begin with XML declaration line as first. In this declaration, the version of XML to which the document conforms must be specified. Parsed data is made up of characters which forms either character data or markups. Markup is used to describe the start tags, end tags, empty elements, entity references, character references, comments, Character Data (CDATA) sections, Document Type Declarations (DTD), Processing Instructions (PI), XML declarations, text declarations, white space characters etc. Markup text is enclosed between the angle brackets < and >. Almost all characters except &, <, >, ',” may be used in an XML document. To represent such exceptional characters, XML uses built-in entities like &amp;, &lt;, &gt; etc. These special characters provide a specific meaning to the application. It is useful when processing the documents.

While processing takes place, the XML document is represented as a tree. In the sample document shown in Figure 2.1, <course> is the root node present at level 0. Two <student> nodes are present at next level as child nodes of <course>. The <student> node has two child nodes <name> and <mark>.

All XML starting tags must have a corresponding ending tag. Proper nesting of tags is needed. Attribute are nested within the element’s
starting tag. Attribute values are enclosed in either single or double quotes. XML is case sensitive. The whitespace is not allowed in the name of XML element or attribute. For example, in the sample document, if the attribute “id” with the value 90 is added to <course> element, then the element becomes <course id="90">. Any number of attributes can be added in an element.

CDATA section allows the user to include data which is not intended to be parsed. Suppose the following CDATA section is included in the sample document “course.xml”.

<![[CDATA[
 If (a>b) then greater:=a;
 Else greater:=b;
 ]]>}

The CDATA section is not parsed. Any text, except the reserved characters like <, > etc can be used inside the CDATA section. Remember that, the sequence “[]]>” also cannot be used inside it. “]]>” is generally used to terminate a CDATA section.

The keyword Parsable Character Data (PCDATA) specifies that the element must contain parsable character data. Depending on the parser and the structure of XML document, insignificant whitespace characters may be collapsed into a single one. Few parsers may remove all such characters fully. This process is called normalization. In the following element, white spaces are present between the words.

<title> XML and Webservices</title>
While parsing, the parser can normalize these whitespaces by collapsing whitespaces into a single significant space. The data becomes “XML and Webservices”.

While creating XML documents, the users can use PI to provide specific data within the XML document. The information in the PI is passed to the application by the parser, while processing that XML document. The PIs are delimited by <? and ?>. PIs consist of PI target and PI value. The users can create their own PIs. But, the reserved words cannot be used as PI target.

While integrating XML documents, two or more elements from different documents may have the same name. This leads to naming collisions. For example, consider another sample document “books.xml”. It has <author> elements as child element. <author> element has <name> and <title> as child elements.

<books>
<author>
   <name > Jack</name>
   <title> Java Programming</title>
</author>
......
</books>

Then, <name> element in “books.xml” refers to the author of the book. But, in the sample document “course.xml”, <name> element refers to the name of a student. If both these elements are used in a single document, there would be a naming collision. Also, it is difficult to determine which kind of data each element contained.
Namespaces are used to prevent the naming collisions. Namespace prefixes are used to specify the namespace in which the element or attribute can be found. Generally, each namespace prefix can be a sequence of characters which is tied to a Uniform Resource Identifier (URI). It uniquely identifies the namespace. The above two <name> elements can be differentiated by prepending namespace prefixes as, <bk:name> Jack</bk:name> and <cs:name>Joe</cs:name>. ‘bk’ and ‘cs’ are namespace prefixes.

Like HTML, comments can be embedded between <!-- and -->. For example, the name of the document can be included as <!-- Sample document: course.xml --> in the sample document.

2.2 PARSING XML

The structure of the XML can be described with the help of XML Schema description languages. This section presents the description about the validation of XML document using these languages. An XML document is considered well formed if it is syntactically correct. XML syntax requires a single root, a start tag and end tag for each element, properly nested tags and attribute values in quotes. The users can impose constraints on storage layout and logical structure.

2.2.1 Document Object Model (DOM) Parser

The W3C provides a standard recommendation for processing XML document. It builds a tree structure in memory which contains XML document’s data. It allows the programmer to manipulate and query the data. A parser that adheres to this W3C recommendation is known as DOM-based parser (W3C 2000). A node in DOM tree may be any one of the elements, attributes etc in XML document. DOM parsers exist for many programming languages like Java, C++ etc. Microsoft’s msxml and Sun Microsystem’s Java API for XML Parsing (JAXP) are examples of DOM-based parsers.
2.2.2 Simple API for XML (SAX) Parser

A SAX based parser processes the document and generates events when XML entities are encountered. While using SAX parser, if it encounters tags, text, comments etc., it gives the notifications to the processing applications. Data from XML document is returned from such events. It does not create any tree structure to store the XML document’s data. This reduces the overhead of using memory. Thus, it has greater performance than DOM parser. These can be used for reading the document which does not need any modification.

2.2.3 XML Schema Description Languages

Structure of an XML document can be defined using either by DTD or XML Schema. DTD define the structure of the document. It specifies what elements, attributes, etc. are permitted in the XML document. The DTD can point to declarations that are inside/outside the XML document which are called as internal/external subset. A document type declaration is placed in the XML document’s prolog. The internal subset for the sample document “course.xml” includes the statement as shown in Figure 2.2.

```xml
<!ELEMENT course(student)*>  
<!ELEMENT student(name, marks)>  
<!ELEMENT name (#PCDATA)>  
<!ELEMENT marks(#PCDATA)>
```

Figure 2.2 DTD of “course.xml”

The user can define the order and the frequency of child elements using DTD of the document. The frequency of the element can be specified by using either the plus sign (+), asterisk (*) or question mark (?) occurrence indicator. An attribute list for an element is declared using the ATTLIST
element type declaration. Entity attributes indicate that an attribute has an entity for its value and are specified using tokenized attribute type ENTITY. Many users feel DTDs are not flexible enough to meet the current programming needs. DTDs cannot be manipulated such as searching, transforming into different representations etc in the same manner as XML documents can. DTD of an XML document does not follow the XML syntax. Also, it cannot validate the content of the document. Data type of the content cannot be specified by using DTD.

XML schemas are an alternative to validating XML documents. Both DTDs and schemas require a validating parser. An XML document that conforms to its schema is considered valid whereas that does not conform is invalid. Any extension may be used for schema documents. The major schema models are W3C XML Schema and Microsoft XML Schema. MicroSoft’s schema document commonly uses the .xsd extension and the schema for the sample document is shown in Figure 2.3.

```xml
<?xml version="1.0"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="course" content="mixed" order="many" model="closed">
    <element type="student"/>
  </ElementType>

  <ElementType name="student" content="eltOnly" order="any"
    model="closed">
    <element type="name"/>
    <element type="marks"/>
  </ElementType>

  <ElementType name="name" content="textOnly"
    model="closed" dt:type="string"/>
  <ElementType name="marks" content="textOnly"
    model="closed" dt:type="int"/>
</Schema>
```

Figure 2.3 Microsoft XML schema “course.xsd” of “course.xml”
In Microsoft XML schema document, `<Schema>` is the root element. The element `<ElementType>` is used to define the XML nodes. `<element>` is used for describing the child elements under `<ElementType>`. To specify the namespace the keyword xmlns is used. In the above schema, “xmlns=urn:schemas-microsoft-com:xml-data” and “xmlns:dt=urn:schemas-microsoft-com:datatypes” are the namespaces to define elements and datatypes respectively. Attribute content specifies the nature of the content allowed in an element or attribute. The content(“eltOnly”) allows only the elements. If it is “textOnly”, it allows only text. Both elements and text will be allowed if the content is “mixed” which is used as default value. The data type of content of an element can also be specified by using the attribute “dt:type”. The attribute order is used to specify the order in which child elements must occur. If order is one, only one child element is permitted. If order is “seq”, then the child elements must appear in the order in which they are defined. If order is “many”, then the child elements can appear in any order, any number of times. It helps the user to insert any number of child nodes in any order. This makes an XML document a semi-structured one.

### 2.2.4 Validating XML

XML parser is a program which is used to read XML documents, checks its syntax and provides access to their content and structure. It also reports any errors in the document. Parsers are classified as validating and non-validating. A validating parser reads the DTD or Schema and determines whether or not the XML document conforms to it. A document is said to be a valid one, if it conforms to its DTD. If it fails to conform to the DTD, but it follows XML syntax, it is known as well-formed but not valid. All valid documents are well-formed. But, the converse is not true. A non-validating parser can read DTD, but cannot check the document against the DTD for conformity.
Parsers can support the either DOM or SAX. Few parsers support both. The users can access document’s content by writing programs using languages such as Java, Python, C etc. DOM Parser builds a tree structure containing the data of the XML document in memory. While using SAX parser, it processes the given document and generates events when tags, text, comments etc are encountered. Microsoft’s msxml, Apache XML Project’s Xerces (Xerces 2005), Sun Microsystem’s Java API for XML Parsing (JAXP), and IBM’s XML for Java (XML4J) are examples of XML parsers. Internet Explorer 5 has built–in XML parser. Validating processors must report violations of the constraints defined by the declarations in the DTD.

2.3 QUERYING XML

Like traditional databases, the XML documents can be queried to get the information from the document (Gou 2005). XML syntax does not provide a way to locate specific pieces of structured data within a given document. The XML Path Language (XPath) (Mary 2004) is used for locating specific parts of an XML document effectively and efficiently.

2.3.1 XPath Expressions

In XPath (Mary 2004), an XML document is conceptually viewed as a tree in which each part of the document is represented as a node. Seven node types are available in XPath. They are root, element, attribute, text, comment, processing instruction and namespace. The result of XPath is a hierarchy of nodes that represent the elements of an XML document in a searchable structure. The XPath tree has a single root node, which contains all other nodes in the tree. The root node and element nodes contain ordered lists of child nodes. The child nodes may be element, comment, text and processing nodes. The relationship between a parent node and a child node is containment, that is, a parent node contains a child node. Each XPath tree
node has a string representation known as “string-value” that XPath uses for comparing the nodes. The XPath expressions may be used with XML Style Sheet and XML Query. For the sample document “course.xml”, few simple XPath expressions are,

```
/course/student - to select all student elements
/course/student/name - to select names of all students in a course
/course/student/marks - to select marks of all students in a course
```

### 2.3.2 XSL Transformation Language (XSLT)

The users use XPath string functions in XSL Style Sheet (XSL) (James Clark 1997) to locate certain nodes in XML document. XSLT(W3c 1999) is used to transform an XML document from one form to another. XSLT uses XPath to match nodes for transforming an XML document into another format. The result of transformation may be an XML, HTML, plain text or any text based document. To transform XSLT documents, an XSLT processor is needed. Microsoft’s Internet Explorer 5 and Apache’s Xalan are examples of XSLT processors. Xalan processor is used with the programming languages like Java and C++.

### 2.3.3 XQuery

Query processing on XML (W3C 2010) needs to determine structural relationships between elements. The result of XML query may be a single element, a group of elements, sub trees etc. Thus, number of challenging research issues are in managing and querying XML data (Kaelin 2004), (Tatarinov 2002). XML Query Language uses the power of XSL patterns to search XML documents for specific data.