Chapter 7. XML

Table of Contents

Objectives

At the end of this chapter you will be able to:

- Explain different markup languages;
- Parse and process XML;
- Use the various XML attributes.

7.1 Introduction to Markup Languages

XML (eXtensible Markup Language) is a markup language for documents that contain structured information. *Markup* refers to auxiliary information interspersed with text to indicate structure and semantics. *Documents* does not only refer to traditional text-based documents, but also to a wide variety of other XML *data formats*, including graphics, mathematical equations, financial transaction over a network, and many other classes of information.

Examples of markup languages include LaTex, which uses markup to specify formatting (e.g. \emph), and HTML which uses markup to specify structure (e.g. <p>). A markup language specifies the syntax and semantics of the markup tags.

Here is a comparison between plain text and marked up text:

Plain text:

```
The quick brown
                fox jumped over the lazy dog.
```

Marked up text:

```
*paragraphstart*The *subjectstart*quick brown fox
      *subjectend* *verbstart*jumped*verbend* over the *objectstart*
      lazy dog*objectend* .*paragraphend*
```

Marked up text aids in semantic understanding, since more information is associated with the sentence than just text itself. This also makes it possible to automatically (i.e. by computer) translate to other formats.

7.1.1 SGML

SGML (Standard Generalised Markup Language) specifies a standard format for text markup. All SGML documents follow a Document Type Definition (DTD that specifies the document's structure). Here is an example:

```
<!DOCTYPE uct PUBLIC "-//UCT//DTD SGML//EN">
<title>test SGML document
<author email='pat@cs.uct.ac.za' office=410 lecturer>Pat Pukram
<version>
   <number>1.0
</version>
```

To do: SGML

Can you see why SGML does not require end tags? Find out more about SGML on the Internet. As a starting point, look at the SGML resources page on the W3 Consortium website [http://www.w3.org/MarkUp/SGML/]. Also search for SGML on Google [http:// www.google.com

7.1.2 HTML

HTML (HyperText Markup Language) specifies standard structures and formatting for linked documents on the World Wide Web. HTML is a subset of SGML. In other words, SGML defines a general framework, while HTML defines semantics for a specific application.

To Do: HTML

HTML is used to specify both the structure and the formatting of Web documents. Examine the list of HTML tags that you have learnt so far and decide which group each tag belongs into. Read up more on the HTML section of the W3 page [http://www.w3.org/MarkUp/].

7.1.3 XML

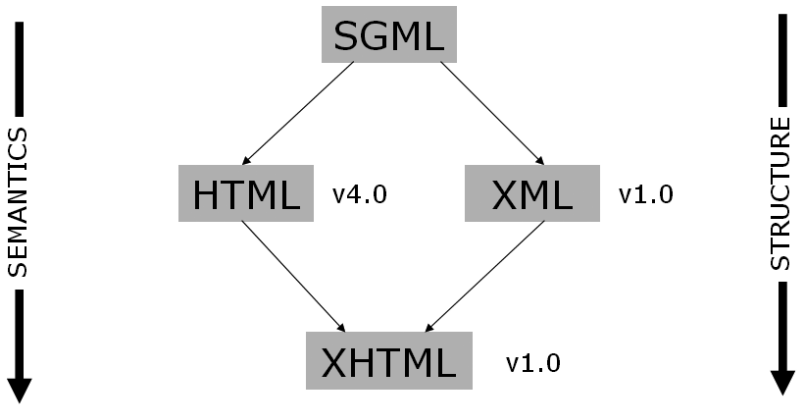
XML, a subset of SGML, was introduced to ease adoption of structured documents on the Web. While SGML has been the standard format for maintaining such documents, its suitability for the Web was poor (for various technical reasons, some of which are discussed later; however, some are beyond the scope of this course). SGML conformity means that XML documents can be read by any SGML system. However, the upside of XML is that XML documents do not require a system capable of understanding the full SGML language.

Both HTML and SGML were considered unsuitable for the use that XML was put to. HTML specifies the semantics of a document (which in HTML's case denotes formatting), but does not provide arbitrary structure. SGML however, does provide arbitrary structure, but is too complex to implement in a Web browser. XML was not designed to replace SGML. As a result, many companies use a SGML to XML filter for their content.

```
<uct>
<title>test XML document</title>
<author email="pat@cs.uct.ac.za" office="410" type="lecturer">Pat
Pukram</author>
<version>
   <number>1.0</number>
</version>
</uct>
```

7.1.4 Relationship

The figure below illustrates the relationship between SGML, HTML, XML, and XHTML. XHTML is discussed later in the chapter.



7.2 XML Primer

An XML document is a serialised segment of text which follows the XML standard (which can be found at the W3C site [http://www.w3.org/TR/REC-xml]).

To Do: Goals of XML

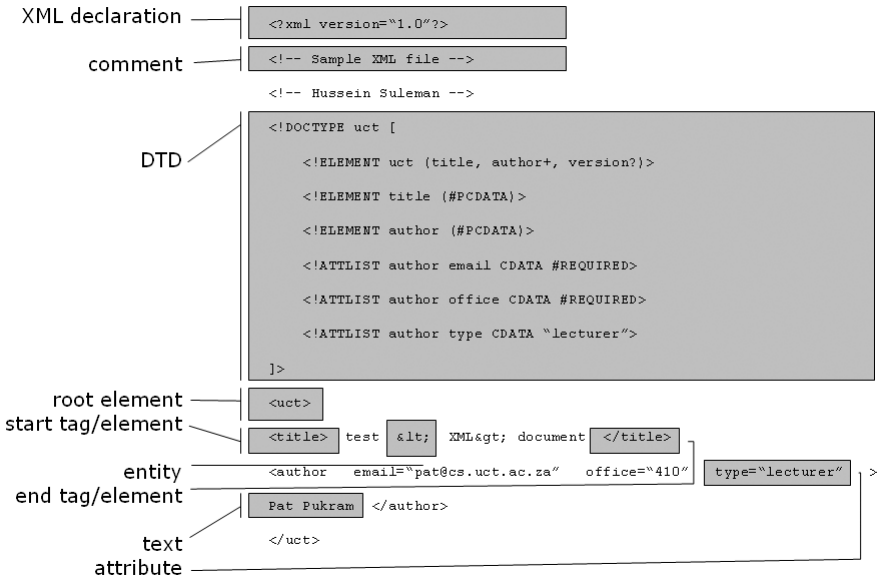
XML's goals are set out in the W3C recommendations [http://www.w3.org/TR/REC-xml/#sec-origin-goals]. Read these recommendations and, if some points are unclear, find out more about them.

An XML document may contain the following items:

- a XML declaration
- DTDs
- text

- elements
- processing instructions
- comments
- entity references

The following image contains an example:



7.2.1 Validity and Well-Formedness

Well-formed XML documents have a single root element, and start and end tags are properly matched and nested. Valid XML documents strictly follow a DTD (or other formal type definition language). Well-formedness enforces the fundamental XML structure, while validity enforces domain- specific structure. SGML parsers, in contrast, have no concept of well-formedness, so domain-specific structure has to be incorporated into the parsing phase.

To Do: Why Validate XML documents

Why do you think it is important to validate XML documents? Discuss this with other students on the online forum.

7.2.2 XML Declaration

The XML declaration appears as the first line of an XML document. Its use is optional. An example declaration appears as follows:

```
<?xml encoding="UTF-8" version="1.0" standalone="yes" ?>
```

encoding indicates how the individual bits correspond to a character set. See the next section for more detail.

version indicates the XML version.

standalone indicates whether an external type definitions must be consulted in order to correctly process the document.

7.2.3 Encoding: Unicode

The encoding used in the above example, UTF-8, is a Unicode-based encoding scheme. Most XML documents are encoded in the ISO 10646 Universal Character Set (also known as UCS or Unicode). Unicode at first supported 16-bit

characters, as opposed to ASCII's 8-bits — this 16-bit format could encode 65536 different characters, taken from most of the known languages. This has since been expanded to 32 bits. The simplest encoding mapping this to 4 fixed bytes is called UCS-4. To represent these characters more efficiently, variable length encodings are typically used instead: UTF-8 and UTF-16.

UTF-16

The Basic Multilingual Plane (characters in the range 0-65535) can be encoded using 16-bit words. Endianness is indicated by a leading Byte Order Mark (BOM) e.g., FF FE = little endian. For more than 16 bits, characters can be encoded using pairs of words and the reserved D800-DFFF range.

```
D800DC00 = Unicode 0x00010000 D800DC01 = Unicode 0x00010001 D801DC01 =
Unicode 0x00010401 DBFFDFFF = Unicode 0x0010FFFF
```

To match UTF-16 to UCS-4:

```
D801-D7C0 = 0041,
DC01 & 03FF = 0001
(0041 << 10) + 0001 = 00010401
```

UTF-8

UTF-8 is optimal for encoding ASCII text, since the first 128 characters needs only 8 bits to encode. Subsequent characters can be encoded using variable encoding. Here are some examples:

```
Unicode 7-bit  = 0vvvvvvv
Unicode 11-bit = 110vvvvv 10vvvvvv
Unicode 16-bit = 1110vvvv 10vvvvvv 10vvvvvv
Unicode 21-bit = 11110vvv 10vvvvvv 10vvvvvv 10vvvvvv etc.
```

Note that the first bits (until the first 0) are used to indicate how many bytes (set of 8 bit) are used to encode the character. Subsequent bytes for the same character encoding begin with 10. The data bits follow each of these header bits (represent by v's in the above examples) in each byte.

To match UTF4 to UTF-8:

```
0001AB45 = 11010 101100 100101
11110vvv 10vvvvvv 10vvvvvv 10vvvvvv
= 11110000 10011010 10101100 10100101
= F09AACA5
```

Note that UTF-8, like UTF-16, is self-segregating to detect code boundaries and prevent errors.

7.2.4 Document Type Definition (DTD)

The Document Type Definition (DTD) defines the structure of an XML document. Its use is optional, and it appears either at the top of the document or in an externally referenced location (a file). Here is an example of a DTD:

```
<!DOCTYPE uct [
    <!ELEMENT uct (title, author+, version?)>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT author (#PCDATA)>
    <!ATTLIST author email CDATA #REQUIRED>
    <!ATTLIST author office CDATA #REQUIRED>
    <!ATTLIST author type CDATA "lecturer">
    <!ELEMENT version (number)>
    <!ELEMENT number (#PCDATA)>
]>
```

ELEMENT defines the basic units of the document's structure. In the above example, they are used to specify different elements (and sub-elements) of documents of type *uct*. The brackets () are used to specify either:

- a list of child elements (sub-element). Each entry in the list can optionally be followed by a symbol each with a different meaning:
- '+': Parent element must have one or more of this child element.
- '*': Parent element must have zero or more of this child element.
- '?': The existence of child element is optional.

In the example above the *uct* element must consist of a *title* element, at least one *author* element and it can optionally contain a *version* element.

- The data type of the leaf-level element (i.e. elements with no children). In the above example, the *title* element is of type *PCDATA* (text). Alternatively the element could consist of an attribute list, which can be defined using the keyword *ATTLIST* in the following way:

```
<ATTLIST parent_element attribute_name attribute_type (#REQUIRED) ("default_v
```

#REQUIRED is optional and can be used to indicate that the attribute is required. *default_value* is also optional and can be used to specify default value for that attribute. In the above example, the *author* element consists of multiple attributes; namely *email* (required), *office* (required) and *type* (this defaults to "*lecturer*" if one was not specified).

Activity 1: DTD

Create a DTD for the following structure:

- **element**: id_data
- **element**: name
- **element**: firstname
- **element**: middlename (0 or more)
- **element**: lastname
- **element**: date of birth
- **required attribute**: day
- **required attribute**: month
- **required attribute**: year
- **element**: bloodgroup (optional)

You can find the solution at the end of the chapter.

7.2.5 Elements / Tags

All elements are delimited by < and >. Element names are case-sensitive and cannot contain spaces (the full character set can be found in the specification). Attributes can be added as space-separated name/value pairs with values enclosed in quotes (either single or double quotes).

```
<sometag attrname="attrvalue">
```

Structure

- Elements may contain other elements in addition to text.
- Start tags begin with "<" and end with ">".
- End tags begin with "<" and end with ">".
- Empty tags (i.e. tags with no content, and the start tag is immediately followed by an end tag) can alternatively be represented by a single tag. These empty tags start with "<" and end with "/>". In other words, empty tags are shorthand. For example: *

* is the same as *
*. This means that, when converting HTML to XHTML, all *
* tags must be in either of the allowed forms of the empty tags.
- Every start tag must have an end tag and must be properly nested. For example, the following is not well-formed, since it is not properly nested.

  ```
  <x><a>mmm<b>mmm</a>mmm</b></x>
  ```

  The following is well-formed:

  ```
  <x><a>mmm<b>mmm</b></a><b>mmm</b></x>
  ```

To Do

Most modern HTML browsers are able to successfully process improperly nested documents. Is this part of the HTML specification? Try to find out more about the similarities and differences between XML and HTML tags.

Special Attributes

An element tag may indicate additional properties for its contents. For example, *xml:space* is used to indicate if whitespace is significant. In general, it is assumed that all whitespace outside of the tag structure is significant. Another special attributes is *xml:lang* which can be used to indicate the language of the content. For example:

```
<p xml:lang="en">I don't speak</p> Zulu
<p xml:lang="es">No hablo</p> Zulu
```

7.2.6 Entities

Entities begin with '&' and end with ';'. Entities represent (refer to) previously defined textual content, usually defined in a DTD. For example, *©* can only be used if the *ISOLat1* entity list is included. Character entities can be used to refer to Unicode characters. For example, *&#23;* refers to decimal character number 23 and *A* refers to hex character number 41. Entities can also refer to predefined escape sequence entities such as < (<), > (>), ' ('), " (") and & (&).

7.3 Creating your own ML based on XML

Relational data, such as the set below, can be encoded into XML format as follows:

class	CS&614	
students	3	
marks	vusi	12

john	24
nithia	36

This data could be encoded as shown below, although there are many other ways of doing this. Can you rewrite it differently?

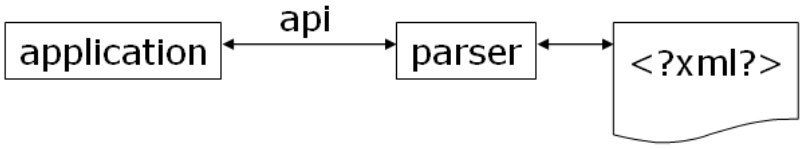
```
<class>CS&amp;614</class>
<students>3</students>
<mark>
  <student_name>vusi</studentname>
  <mark_obtained>12</mark_obtained>
</mark>
<mark>
  <student_name>john</student_name>
  <mark_obtained>24</mark_obtained>
</mark>
<mark>
  <student_name>nithia</student_name>
  <mark_obtained>36</mark_obtained>
</mark>
```

Now encode the following in XML:

Name	**Age**	**Occupation**
Tsepo	20	Student
James	19	Waiter
Molly	27	Executive

7.4 Parsing and Processing XML

XML parsers process both the data contained in an XML document, as well as the data's structure. In other words, they expose both to an application, as opposed to regular file input where an application only receives content. Applications manipulate XML documents using APIs exposed by parsers. The following diagram show the relationship.



Two popular APIs are the Simple API for XML (SAX) and Document Object Model (DOM).

7.4.1 SAX

The Simple API for XML (SAX) is an event-based API that uses callback routines or event handlers to process different parts of an XML documents. To use SAX, one needs to register handlers for different events and then parse the document. Textual data, tag names and attributes are passed as parameters to the event handlers.

To Do

Read up about SAX on the Internet. A good page to start is the SAX project home page [http://sax.sourceforge.net/].

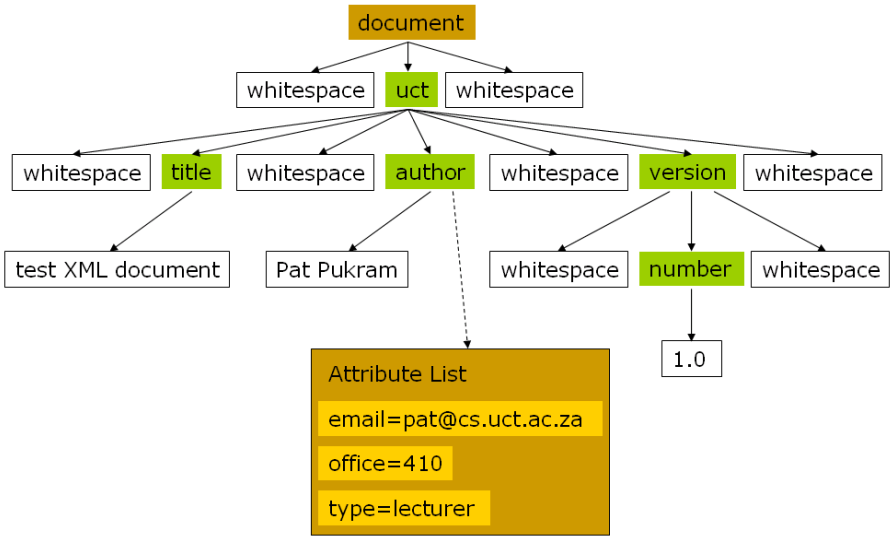
Using handlers to output the content of each node, the following output can be trivially generated:

```
start document start tag : uct start tag : title
content : test XML document end tag : title
start tag : author content : Pat Pukram end tag : author start tag :
version start tag : number content : 1.0
end tag : number end tag : version end tag : uct
end document
```

7.4.2 DOM

The Document Object Model (DOM) defines a standard interface to access specific parts of the XML document, based on a tree-structured model of the data. Each node of the XML document is considered to be an object with methods that may be invoked to get/set its contents/structure, or to navigate through the tree. DOM v1 and v2 are W3C [www.w3c.org] standards with DOM3 having become a standard as of April 2004.

Here is a DOM tree of our example:



You might be able to understand the following code. Perl is a popular language to use in DOM processing because of its text-processing capabilities. Java is also popular because of its many libraries and Servlet support.

Step-by-step Parsing

```
# create instance of parser my $parser = new DOMParser;
# parse document
my $document = $parser->parsefile ('uct.xml');
# get node of root tag
my $root = $document->getDocumentElement;
# get list of title elements
my $title = $document->getElementsByTagName ('title');
# get first item in list
my $firsttitle = $title->item(0);
# get first child – text content
my $text = $firsttitle->getFirstChild;
# print actual text print $text->getData;
```

Quick-and-dirty Approach

```
my $parser = new DOMParser;
my $document = $parser->parsefile ('uct.xml');
print $document->getDocumentElement->getElementsByTagName ('title')->item(0)->g
```

DOM Interface Subset

Different level of the DOM tree have different attributes and methods:

- **Document**
- **attributes**: documentElement
- **methods**: createElement, createTextNode, ...
- **Node**
- **attributes**: nodeName, nodeValue, nodeType, parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling, attributes
- **methods**: insertBefore, replaceChild, appendChild, hasChildNodes
- **Element**
- **methods**: getAttribute, setAttribute, getElementsByTagName
- **NodeList**
- **attributes**: length
- **methods**: item
- **CharacterData**
- **attributes**: data

DOM Bindings

The DOM has different bindings in different languages. Each binding must cater for how the document is parsed — this is not part of DOM. In general, method names and parameters are consistent across bindings. Some bindings define extensions to the DOM, for example, to serialise (turn into a linear data structure) an XML tree.

To Do

Read up more about DOM on the Internet. You can start by looking at the W3C's section on DOM [http://www.w3.org/DOM/]. Also make use of search engines.

7.4.3 SAX vs DOM

Below we compare SAX and DOM.

- DOM is a W3C standard while SAX is a community-based "standard".
- DOM is defined in terms of a language-independent interface, while SAX is specified for each implementation language (with Java being the reference).
- DOM requires reading the whole document to create an internal tree structure while SAX can process data as it is parsed. In general, DOM uses more memory to provide random access.

7.5 XML Namespaces

Namespaces partition XML elements into well-defined subsets in order to prevent name clashes between elements. If two XML DTDs define the tag "title", which one is implied when the tag is taken out of its document context (e.g., during parsing)? Namespaces disambiguate the intended semantics of XML elements.

7.5.1 Default Namespaces

If no namespace is specified for an element, it is placed in the default namespace. An element's namespace (and the namespace of all of its children) is defined with the special "*xmlns*" attribute on an element. Example:

```
<uct xmlns="http://www.uct.ac.za">
```

Namespaces are specified using URIs, thus maintaining uniqueness. Universal Resource Locator (URL) = location-specific

Universal Resource Name (URN) = location-independent Universal Resource Identifier (URI) = generic identifier

7.5.2 Explicit Namespaces

Multiple active namespaces can be defined using prefixes. Each namespace is declared with the attribute "*xmlns:ns*", where *ns* is the prefix to be associated with the namespace. The containing element and its children may then use this prefix to specify their membership to a namespace other than the default.

```
<uct xmlns="http://www.uct.ac.za" xmlns:dc="http://somedcns">
            <dc:title>test XML document</dc:title>
</uct>
```

7.6 XML Schema

A XML Schema is an alternative to the DTD for specifying an XML document's structure and data types. It is capable of expressing everything a DTD can, and more. Similar, alternative languages exist, such as RELAX and Schematron, but XML Schemas are a W3C standard.

7.6.1 Schema Structure

Elements are defined using <element name="..." type="..." minOccurs="..." maxOccurs="...">, where:

- *name* refers to the tag.
- *type* can be custom-defined or one of the standard types. Common predefined types include *string*, *integer* and *anyURI*.
- *minOccurs* and *maxOccurs* specify how many occurrences of the element may appear in an XML document. *unbounded* is used to specify no upper limits.

Example: <element name="title" type="string" minOccurs="1" maxOccurs="1"/>

7.6.2 Sequences

Sequences of elements are defined using a complexType container:

```
<complexType>
   <sequence>
      <element name="title" type="string"/>
      <element name="author" type="string" maxOccurs="unbounded"/>
   </sequence>
</complexType>
```

Note: Defaults for both minOccurs and maxOccurs are 1.

7.6.3 Nested Elements

Instead of specifying an atomic type for an element, its type can be elaborated as a structure. This corresponds to nested XML elements.

```
<element name="uct">
   <complexType>
      <sequence>
         <element name="title" type="string"/>
         <element name="author" type="string" maxOccurs="unbounded"/>
      </sequence>
   </complexType>
</element>
```

7.6.4 Extensions

Extensions are used to place additional restrictions on an element's content. For instance, the content can be restricted to be a value from a given set:

```
<element name="version">
   <simpleType>
      <restriction base="string">\
         <enumeration value="1.0"/>
         <enumeration value="2.0"/>
      </restriction>
   </simpleType>
</element>
```

The content can be forced to conform to a regular expression:

```
<element name="version">
   <simpleType>
      <restriction base="string">
         <pattern value="[1-9]\.[0-9]+"/>
      </restriction>
   </simpleType>
</element>
```

7.6.5 Attributes

Attributes can be defined as part of complexType declarations.

```
<element name="author">
   <complexType>
      <simpleContent>
         <extension base="string">
            <attribute name="email" type="string" use="required"/>
            <attribute name="office" type="integer" use="required"/>
            <attribute name="type" type="string"/>
         </extension>
      </simpleContent>
   </complexType>
</element>
```

7.6.6 Named Types

Types can be named and referred to at the top level of the XSD.

```
<element name="author" type="uct:authorType"/>
```

```
<complexType name="authorType">
   <simpleContent>
      <extension base="string">
         <attribute name="email" type="string" use="required"/>
         <attribute name="office" type="integer" use="required"/>
         <attribute name="type" type="string"/>
      </extension>
   </simpleContent>
</complexType>
```

7.6.7 Other Content Models

Instead of sequence, other content models may be used:

- *choice* means that only one of the children may appear.
- *all* means that each child may appear or not, but at most once each.

Consult the specification for more detail on these and other content models.

7.6.8 Schema Namespaces

Every schema should define a namespace for its elements and for internal references to types. For example:

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

<element name="author" type="uct:authorType"/>

<complexType name="authorType">
   <simpleContent>
      <extension base="string">
         <attribute name="email" type="string" use="required"/>
         <attribute name="office" type="number" use="required"/>
         <attribute name="type" type="string"/>
                        </extension>
   </simpleContent>
</complexType>

</schema>
```

7.6.9 Schema Example

Here is an example of a full Schema:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.uct.ac.za"
        xmlns:uct="http://www.uct.ac.za" elementFormDefault="qualified"
        attributeFormDefault="unqualified"
>
<complexType name="authorType">
   <simpleContent>
      <extension base="string">
         <attribute name="email" type="string" use="required"/>
         <attribute name="office" type="integer" use="required"/>
         <attribute name="type" type="string"/>
      </extension>
   </simpleContent>
```

```
</complexType>

<complexType name="versionType">
   <sequence>
      <element name="number">
         <simpleType>
            <restriction base="string">
               <pattern value="[1-9]\.[0-9]+"/>
            </restriction>
         </simpleType>
      </element>
   </sequence>
</complexType>

<complexType name="uctType">
   <sequence>
      <element name="title" type="string"/>
      <element name="author" type="uct:authorType"/>
      <element name="version" type="uct:versionType"/>
   </sequence
</complexType>

<element name="uct" type="uct:uctType"/>

</schema>
```

Here is a valid XML example for the above Schema

```
<uct xmlns="http://www.uct.ac.za"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.uct.ac.za
       http://www.husseinsspace/teaching/uct/2003/csc400dl/uct.xsd"
>

   <title>test XML document</title>
   <author email="pat@cs.uct.ac.za" office="410"
           type="lecturer">Pat Pukram</author>
   <version>
                     <number>1.0</number>
   </version>
</uct>
```

Activity 2: Schema

Write a Schema for the following XML document.

```
   <article xmlns="http://article.com">
     <name>Fermat's Last Theorem</name>
     <date>20010112</date>
     <length unit="pages">11</length>
     <author>
       <first>Jonathan</first>
                    <last>Smith</last>
     </author>
     <author>
       <first>Mary</first>
       <last>Carter</last>
     </author>
   </article>
```

7.7 Data and Metadata

Data refers to digital objects that contain useful information for information seekers. Metadata refers to descriptions of these objects. Many systems manipulate metadata records, which contain pointers to the actual data.

7.7.1 Metadata Standards

To promote interoperability among systems, there are popular metadata standards to describe objects (both semantically and syntactically).

- **Dublin Core**: uses fifteen simple elements to describe every object.
- **MARC**: a comprehensive system devised to describe items in a (physical) library.
- **RFC1807**: the computer science publications format.
- **IMS Metadata Specification**: courseware object description.
- **VRA-Core**: multimedia (especially image) description.
- **EAD**: aids to locate archived items.

Dublin Core Example

Dublin Core is one of the most popular (and simplest) metadata formats. It contains fifteen elements, each with recommended semantics. All the elements are optional and repeatable. They are:

Title	Creator	Subject
Description	Publisher	Contributor
Date	Type	Format
Identifier	Source	Language
Relation	Coverage	Rights

Below is a Dublin Core in XML example:

```
<oaidc:dc xmlns="http://purl.org/dc/elements/1.1/"
xmlns:oaidc="http://www.open
   <title>02uct1</title>
   <creator>Hussein Suleman</creator>
   <subject>Visit to UCT </subject>
   <description>the view that greets you as you emerge from the tunnel
   under th
   <publisher>Hussein Suleman</publisher>
   <date>2002-11-27</date>
   <type>image</type>
   <format>image/jpeg</format>
   <identifier>http://www.husseinsspace.com/pictures/200230uct/02uct1.j
   pg
</identifier>
   <language>en-us</language>
   <relation>http://www.husseinsspace.com</relation>
   <rights>unrestricted</rights>
</oaidc:dc>
```

7.7.2 Metadata Transformation

To do this, take the following steps:

1. Use an XML parser to parse data.

2. Use SAX/DOM to extract individual elements and generate the new format.

The following code converts UCT to Dublin Core (Don't worry if you do not understand it):

```
my $parser = new DOMParser;
my $document = $parser->parsefile ('uct.xml')->getDocumentElement;
foreach my $title ($document->getElementsByTagName ('title'))
{
   print "<title>".$title->getFirstChild->getData."</title>\n";
}
foreach my $author ($document->getElementsByTagName ('author'))
{
   print "<creator>".$author->getFirstChild->getData."</creator>\n";
}
print "<publisher>UCT</publisher>\n";
foreach my $version ($document->getElementsByTagName ('version'))
{
   foreach my $number ($version->getElementsByTagName ('number'))
   {
      print "<identifier>".
            $number->getFirstChild->getData."</identifier>\n";

   }
}
```

As you will see later in this unit, there is an easier way to achieve this in the unit.

7.8 XPath

The XML Path Language (XPath) supplies a mechanism to address particular nodes or sets of nodes in an XML document. XPath expressions can be used to write precise expressions to select nodes without using procedural DOM statements. For example, we can address particular nodes using expressions like:

```
uct/title uct/version/number uct/author/@office
```

7.8.1 XPath Syntax

- Expressions are separated by "/".
- In general, each subexpression matches one or more nodes in the DOM tree.
- Each sub-expression has the form: *axis::node[condition1][condition2]...* where *axis* can be used to select children, parents, descendants, siblings, and so on.
- Symbols may be used for the possible axes:

Expression	**What it selects in current context**
title	"title" children
*	All children
@office	"office" attribute
author[1]	First author node
/uct/title[last()]	Last title within uct node at top level of document
//author	All author nodes that are descendant from top level
.	Context node

..	Parent node
version[number]	Version nodes that have "number"
version[number='1.0']	Version nodes for which "number" has content of "1.0"

7.9 XSL

The XML Stylesheet Language (XSL) converts structured, XML files into a "human-friendly" representation. The thought underlying this is that, besides for programmers, no one should ever have to read or write XML. Conversions can be done in two steps:

1. Transform XML data
2. Process the data and stylesheet.

In systems that are Web-based, the first step is more useful — the first step being called XSL Transformations (XSLT) — as XHTML is directly "processed" by browsers.

7.10 XSLT

XSLT is a declarative language, written in XML, that specifies transformation rules for XML fragments. XSLT can convert any arbitrary XML document into XHTML or another XML format (e.g., different metadata formats). For example, the author tag can be converted as follows:

```
<template match="uct:author">
   <dc:creator>
      <value-of select="."/>
   </dc:creator>
</template>
```

7.10.1 XSLT Templates

Templates of replacement XML are specified, with matching criteria, using XPath expressions. XSLT processors attempt to match the root XML tag with a template. If this fails they descend one level and try to match each of the root's children, and so on. In the previous example, all occurrences of the "uct:author" tag will be replaced by the contents of the template. Special tags in the XSL namespace are used to perform additional customisation, for example, *value-of*.

7.10.2 XSLT Special Tags

- **value-of, text, element**: Create nodes in result document.
- **apply-templates, call-template**: Explicitly apply template rules.
- **variable, param, with-param**: Local variables and parameter passing.
- **if, choose, for-each**: Procedural language constructs.

7.10.3 XSLT Language

value-of is replaced with the textual content of the nodes identified by the XPath expression. For example:

```
<value-of
      select="uct:title"/>
```

text is replaced by the textual content. Plain text is usually sufficient. For example:

`<text>1.0</text>` 1.0

element is replaced by an XML element with the indicated tag. Usually the actual tag can be used. Example:<element name="dc:publisher">UCT</element> <dc:publisher>UCT</dc:publisher>

apply-templates explicitly applies templates to the specified nodes. Example: apply-templates select="uct:version"/>

call-template calls a template in a similar way to calling a function. This template may have parameters and must have a *name* attribute instead of a *match*. Example:<call-template name="doheader"> <with-param name="lines">5</with-param> </call-template> <template name="doheader"> <param name="lines">2</param> ... </template>

variable sets a local variable. In XPath expressions, a $ prefix indicates a variable or parameter instead of a node. Example:<variable name="institution">UCT</variable> <value-of select="$institution"/>

Selection and iteration examples:<if test="position()=last()">...</if> <choose> <when test="$val=1">...</when> <otherwise>...</otherwise> </choose> <for-each select="uct:number">...</for-each>

7.10.4 XSLT Example

```
<stylesheet version='1.0' xmlns='http://www.w3.org/1999/XSL/Transform'
 xmlns:oaidc='http://www.openarchives.org/OAI/2.0/oai_dc/'
 xmlns:dc='http://purl.org/dc/elements/1.1/'
 xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
 xmlns:uct='http://www.uct.ac.za'
>

<!--
   UCT to DC transformation Hussein Suleman
   v1.0 : 24 July 2003
-->

   <output method="xml"/>

   <variable name="institution"><text>UCT</text></variable>
   <template match="uct:uct">
      <oaidc:dc
                xsi:schemaLocation="http://www.openarchives.org/OAI/2.0
                /oai_dc/
                http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
         <dc:title><value-of select="uct:title"/></dc:title>
         <apply-templates select="uct:author"/>
         <element name="dc:publisher">
               <value-of select="$institution"/>
         </element>
         <apply-templates select="uct:version"/>
      </oaidc:dc>
   </template>

   <template match="uct:author">
      <dc:creator>
         <value-of select="."/>
      </dc:creator>
   </template>

   <template match="uct:version">
      <dc:identifier>
```

```
            <value-of select="uct:number"/>
         </dc:identifier>
      </template>

</stylesheet>
```

This is not the simplest XSLT that solves the problem. The transformed XML looks like this:

```
<?xml version="1.0"?>
<oaidc:dc xmlns:oaidc="http://www.openarchives.org/OAI/2.0/oai_dc/"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:dc="http://purl.org/dc/elements/1.1/"
          xmlns:uct="http://www.uct.ac.za"
          xsi:schemaLocation=
             "http://www.openarchives.org/OAI/2.0/oai_dc/
                 http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
  <dc:title>test XML document</dc:title>
                   <dc:creator>Pat Pukram</dc:creator>
  <dc:publisher
  xmlns:dc="http://purl.org/dc/elements/1.1/">UCT</dc:publisher>
                   <dc:identifier>1.0</dc:identifier>
</oaidc:dc>
```

7.11 Answers

7.11.1 Answer to Activity 1

One possible DTD is:

```
<!DOCTYPE id_data [
    <!ELEMENT id_data (name, date_of_birth, blood_group?)>
    <!ELEMENT name (firstname, middlename*, lastname)>
    <!ELEMENT firstname (#PCDATA)>
    <!ELEMENT middlename (#PCDATA)>
    <!ELEMENT lastname (#PCDATA)>
    <!ATTLIST date_of_birth day CDATA #REQUIRED>
    <!ATTLIST date_of_birth month CDATA #REQUIRED>
    <!ATTLIST date_of_birth year CDATA #REQUIRED>
    <!ELEMENT blood_group (#PCDATA)>
]>
```