Assignment 1 - Structured Information

Group # 1

1 XML Schema Basics

Basically our schema for the management of books is divided in two files:

- booksattr.xsd.- It contains all the attributes that belongs to a book. In this scheme, we have used simple type of elements to specifies the constraints and information about the values, such as the restrictions of attributes on the language, number of pages of a book. Besides, it is also used to define if a book includes some resource (e.g.CD,Maps). The xmlns value provide the namespaceURI to the xs schema. The targetNamespace value represents a URI reference of the namespace of this schema. Listing 1 shows the definition of the first part of this scheme.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns="urn:booksattr"
                   targetNamespace="urn:booksattr"
                   xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="author"      type="xs:string"/>
  <xs:element name="title"       type="xs:string"/>
  <xs:element name="genre"       type="xs:string"/>
  <xs:element name="bookshelf"   type="xs:integer" />
  <xs:element name="review" type="xs:string"/>
  <xs:element name="language" >
        <xs:simpleType>
          <xs:restriction base="xs:string">
              <xs:enumeration value="English"/>
              <xs:enumeration value="Finnish"/>
              <xs:enumeration value="Other languages"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
  <xs:element name="npages">
              <xs:simpleType>
                <xs:restriction base="xs:positiveInteger">
                 <xs:maxExclusive value="2000"/>
                </xs:restriction>
               </xs:simpleType>
      </xs:element>
  <xs:element name="resource">
              <xs:simpleType>
          <xs:restriction base="xs:string">
              <xs:enumeration value="CD"/>
              <xs:enumeration value="MAPS"/>
              <xs:enumeration value="NO"/>
          </xs:restriction>
        </xs:simpleType>
     </xs:element>
```

Listing 1: Scheme for elements of a book (a)

Listing 2 shows the use of a complexType element which references the elements previously defined. The complexType element contains other elements and/or attributes such as sequence element and the defined elements and attribute element.

```
    <xs:complexType name="BookForm">
     <xs:sequence>
       <xs:element ref="author"/>
       <xs:element ref="title"/>
       <xs:element ref="genre"/>
       <xs:element ref="bookshelf"/>
       <xs:element ref="language"/>
       <xs:element ref="npages"/>
       <xs:element ref="resource"/>
       <xs:element ref="review"/>
     </xs:sequence>
     <xs:attribute name="id"   type="xs:string"/>
   </xs:complexType>
</xs:schema>
```

Listing 2: Scheme for elements of a book (b)

- books.xsd.- This file imports the schema defined in *bookattr.xsd* by using namespace as well as schemaLocation in order to form the main element *"books"* .

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            targetNamespace="urn:books"
            xmlns:bks="urn:booksattr"
            xmlns="urn:books"
            elementFormDefault="qualified">
  <xs:import namespace="urn:booksattr" schemaLocation="booksattr.xsd"/>

  <xs:element name="books">
   <xs:complexType>
    <xs:sequence>
      <xs:element name="book"
                  type="bks:BookForm"
                  minOccurs="0"
                  maxOccurs="unbounded"/>
      </xs:sequence>
  </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing 3: Schema for books

Once defined the schemes, we proceeded to create the xml file that would have the same structure defined on *books.xsd* in order to be valid. Listing 4 presents how it should looks the declaration of a book in a .xml file.

```
    <book id="bk001">
       <bks:author>Hightower, Kim</bks:author>
       <bks:title>The First Book</bks:title>
       <bks:genre>Fiction</bks:genre>
       <bks:bookshelf>1</bks:bookshelf>
       <bks:language>English</bks:language>
       <bks:npages>100</bks:npages>
       <bks:resource>CD</bks:resource>
       <bks:review>An amazing story of nothing.</bks:review>
    </book>
    <book>....</book>
```

Listing 4: Example of a books based on the scheme

2 Programmatic Approach to XML

The validation and parsing of the files was implemented in Java with the use of two packages:*javax.xml* and *org.xml* as it is presented in [1] and [2] . Also, it was defined a class *Book.java* for better management of the extraction of each attribute of the XML file. Listing 5 presents the basic definition of the class and packages used on this project.
In main JAVA file, we also define two types of argument, namely, *"validate"* and *"list"*

- *"validate"* is used to validate the xml file with the defined xml schema.
- *'list'* is used to list all books with details in the xml file by searching the element node of value 'book'.

```java
package Classes;
import java.io.File;
import org.w3c.dom.*;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

public class Book {
         private int id_book;
         private String author;
         private String title;
         private String genre;
         private int bookshelf;
         private String language;
         private int npages;
         private String resource;
         private String review;
}
```

Listing 5: Definition of book class and packages used

3 Extra Features

In this part we implemented two extra features according to the assignment, namely *'XML to JSON'* and *'Addition And Deletion for Bookshelf'*. *'JSON'* is a data format which is widely used in web application development since it is greatly supported by Javascript.
Firstly, we choose PYTHON as the programming language to change *'XML'* file to *'JSON'* file. Because it is a easy-to-understand modern language and has many popular used libraries. In this case, we use *'xmltodict'* [3] and *'json'* [4] libraries, which is officially marked as 'Makes working with XML feel like you are working with JSON'. The main idea is parse the nodes in the xml file and then dump to the output file by using json library in PYTHON. The code is quite simple and is shown on Listing 6.
Listing 7 shows the generated json file. The attribute of the element is defined begin with *"@"*. Besides, elements and their values are listed in python's dictionary way ('key' and 'value').
Next, we add two types of arguments in main JAVA file, namely *'del'* and *'add'*.
'del' is used to delete a book with its list number, for example, *'del 2'* will delete the 2nd book in the xml

[1] http://www.developerfusion.com/code/2064/a-simple-way-to-read-an-xml-file-in-java/
[2] http://tutorials.jenkov.com/java-xml/dom-schema-validation.html
[3] https://pypi.python.org/pypi/xmltodict
[4] http://docs.python.org/2/library/json.html

```
'''
use xmltodict library to export json file from xml file
'''
import io, xmltodict, json
infile = io.open("books.xml", 'r')
outfile = io.open("books.json", 'wb')
o = xmltodict.parse( infile.read() )
json.dump( o , outfile, indent=2)
```

Listing 6: Code to export from xml file to json file

```
{
  "x:books": {
    "@xmlns:x": "urn:books",
    "@xmlns:xsi": "http://www.w3.org/2001/XMLSchema-instance",
    "@xsi:schemaLocation": "urn:books books.xsd",
    "book": [
      { "@id": "bk001",
        "author": "Hightower, Kim",
        "title": "The First Book",
        "genre": "Fiction",
        "bookshelf": "1",
        "language": "English",
        "npages": "100",
        "resource": "CD",
        "review": "An amazing story of nothing."
      },
    ]  } }
```

Listing 7: JSON File generated

file. Listing 9 shows how *'del'* works. First, the node to be deleted is selected and then we use *streamResult* class, save the output file in the books xml file.
'add' is used to add the books in another file to the main books xml file, (in our case, we use another file named *extrabook.xml* to save the added books). Listing 8 shows how *'Add'* works. Compared to *'del'*, in JAVA, it is not allowed to directly append the new node from another file, instead, the added node need to be imported to a new one and then append to the one in the main file.

```
String path_file = new java.io.File(".").getCanonicalPath() + "/files";
Document doc = docBuilder.parse (new File(path_file+"/books.xml"));
Document doc_add = docBuilder.parse (new File(path_file+"/extrabook.xml"));
doc.getDocumentElement ().normalize ();
doc_add.getDocumentElement ().normalize ();
//GET node from two files
NodeList listBook = doc.getElementsByTagName("book");
NodeList listAddBook = doc_add.getElementsByTagName("book");
int totalBooks = listBook.getLength();
int totalAddBooks = listAddBook.getLength();
System.out.println("The number of books to be added: " + Integer.toString(totalAddBooks) );
//GET last node of books.xml
Node n=listBook.item(totalBooks-1);
//ADD elements in extrabook.xml to books.xml
for(int s=0; s<totalAddBooks ; s++){
        Node m= listAddBook.item(s);
        // APPEND the add node next to the last node of books.xml
        Node imp = doc.importNode(m,true);
        n.getParentNode().appendChild(imp); }
```

Listing 8: Add books from an external file

```
 String path_file = new java.io.File(".").getCanonicalPath() + "/files";
 Document doc = docBuilder.parse (new File(path_file+"/books.xml"));
 doc.getDocumentElement ().normalize ();
 NodeList listBook = doc.getElementsByTagName("book");
 int totalBooks = listBook.getLength();
 if(book_number>totalBooks){
         System.out.println("There is no this book, please input a correct book number again!");
         System.out.println();
     System.exit(0);
 } else{
// GET the remove book node
     Element book = (Element)listBook.item(book_number-1);
     book.getParentNode().removeChild(book);
     System.out.println("Remove the " + Integer.toString(book_number) + "th book");
     // SAVE to the xml file
     TransformerFactory tf = TransformerFactory.newInstance();
     Transformer t = tf.newTransformer();
     DOMSource source = new DOMSource(doc);
     StreamResult streamResult =  new StreamResult(new File(path_file+"/books.xml"));
     t.transform(source, streamResult);
     System.out.println();
     System.out.println("Update the xml file succesfully!"); }
```

Listing 9: Remove the selected book

4 Learning Experience

In overall, the exercise was really useful for us in order to understand how to define a XML schema and also the process of validation by using Java libraries.
The most difficult part of the assignment was the definition of our scheme because we had some points to consider and sometimes it could be arduous to include all the details provided in the description. By the other hand, the most easy part of this assignment was the validation of our scheme because we were familiar with Java and also the uses of some libraries really helped us to pursue the objective.
Each of us have worked about 10 hours to define the schema, write the code for validation and this report.