2014

Network Application Framework

ASSIGNMENT 1: STRUCTUREDINFORMATION

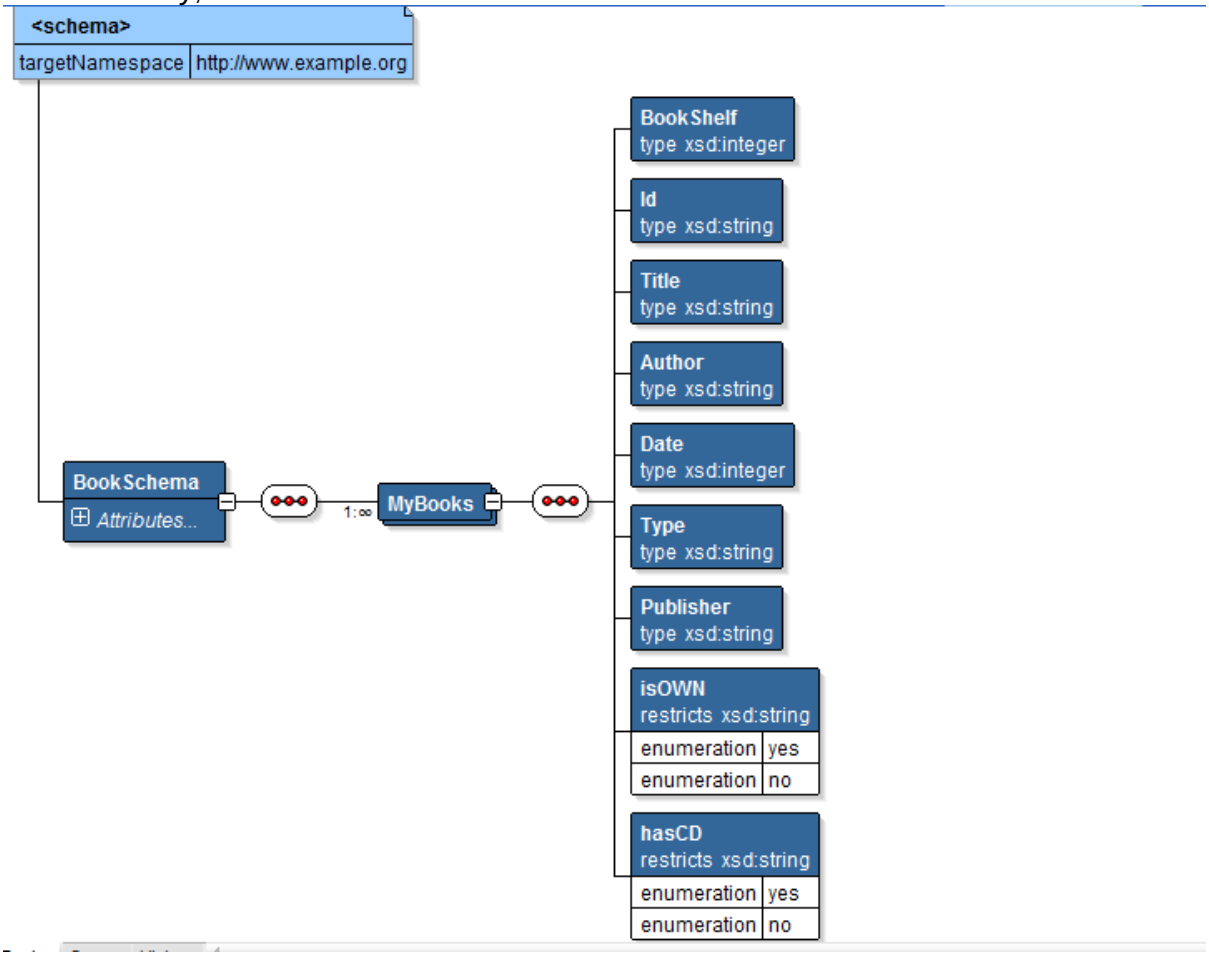
Kiran Kumar	**399876**	**kiran.kumar@aalto.fi**
Radhakrishnan Ilavarasi	**400613**	**ilavarasi.radhakrishnan@aalto.fi**

1. Our Bookshelf Story

I am an unorganised person having 2 book shelves at my place. All my technical books and leisure books are mixed up in both the book shelves. I have a few text books related to computers and networking. A few of the books have CD's in them. I have borrowed quite a few books from my friends and library. There are a few detective novels and some magazines. I believe I have seen some comics in one of the shelf. Since I'm studying abroad, I have a few cook books to learn and cook a few of my favourite dishes.

2. XML Schema

After the story, we listed out some of the common attributes of books.



The schema consists of a complex type BookSchema and inside that another complex type MyBooks which has 1:Many mapping (i.e. under the complex type BookSchema, their might be many MyBooks element).As there are lot of different kinds of books, the MyBook element's all the attributes might not be applicable to all kinds of books. Hence the minor ccurs of all the attributes is not given in the schema. The isOWN attribute describes wheather the book is borrowed and the hasCD signifies the presence of CD with the book. Both the attributes has been defined as list of values and can have only values 'yes' or 'no'. The date is defined as a string as some publications have only year and

date format cannot be specified. The first attribute BookShelf signifies weather the book is present in the 1st or 2nd book shelf.

3. Programmatic Approach to XML

The main method is present in xmlValidateList.java, which defines the functions validate and list.

An initial check is made on the argument provided in the command line to verify if the arguments given are valid. Try and catch are used to handle exceptions. This main method has an IF – ELSE construct to see if the arguments passed in the command line is list or validate.

If the argument is validate -

The `validateXMLSchema` method will validate the xml file against the schema. The validate function will check if the xml file is according to the correct format with respect to the schema described. If it's valid it says the "validation is true". If the xml does not associate with the schema "validation is false". This feature is provided by the following imports:

- `import javax.xml.validation.Schema;`
- `import javax.xml.validation.SchemaFactory;`
- `import javax.xml.validation.Validator;.`

If the argument is list-

The main method first calls the validate function. If the validate function returns true, then it lists all the values in the XML using DOM parser. Here javax.xml.parsers package is used to implement the parsing. This feature is provided by the following imports:

- `import javax.xml.parsers.DocumentBuilderFactory;`
- `import javax.xml.parsers.DocumentBuilder;`
- `import org.w3c.dom.Document;`
- `import org.w3c.dom.NodeList;`
- `import org.w3c.dom.Node;`
- `import org.w3c.dom.Element;`

We have implemented error handling for each and every block of code to display exact error messages during execution.

4. Usage Instructions

The code can be executed in 2 methods:

First Method:

1. Open the project in Eclipse IDE
2. Find the location of the xml file you need to validate (Eg: /Desktop/xxx.xml)

3. Open Run Configurations window in Eclipse and provide the arguments in the following usage format --- <filename with location> <operation/function> and run the code
4. Output can be seen in the console window

Second Method:

1. Open command line
2. Check if all the files(booklist.xml,bookshelf.xsd and xmlValidateList.java) are the same directory.
3. Compile the java program with the following command:

javac xmlValidateList.java

4. Run the program: The required format is

java <java filename> <xmlfilename> <option=validate or list>

Hence here the program can be run as

java xmlValidateList New1.xml validate

5. Learning Experience

First we tried to understand the problem statement. An initial read on how XML works and basic syntax of XML helped us a lot in understanding the assignment. Then we came up with the bookshelf story with few negotiations between us. Then after having a rough idea of our bookshelf story, we tried to write xsd file. Following this we started with the xml file where we faced little difficulty in writing the target space, namespace syntax. Then we implemented the java function validate and the required. We then tested it for the working of the same. We had to learn on different out of the box parsing functions that Java provided.

We reserve some time studying the operation and the method of resolution of this problem using different JAVA functions, finally we came up with a solution of using DOM parser to parse the XML for validation and listing. We implemented the code using Eclipse IDE and tested the same code in command line in Ubuntu When the code was finally working properly, we started testing the code with different kinds of XML files to test the error handling functionality.

This assignment helped us understand better the working of XML messaging, XPATH and JAVA inbuilt functions to handle XML's.

5.1 Easy tasks

Starting with the story was bit easy and the xml file was easy.

5.2 Difficult tasks

As said before, understanding target name, namespace (i.e providing the name url in all the places (3) were initially confused by us then we got to know it

better. Also we were bit confused about the validation part. We were unsure if the xml need to be validated against schema or do we need separate database where the book details will be stored and then this database to be validated against xml schema.

6. Workload per person

Working hours per person:

As a team the same amount of time was invested by both of us. Task and the approximate time taken in hours as follows:

Xml and xsd studying: **5**
Story definition: **1**
Xml file: **1**
Xsd file: **1**
Validate function: **8**
List function: **1**
Studying related to the bug faced: **8**
Final changes in the code overall: **2**
Report: **2**

TOTAL: **29** hours