# Writing an XML Schema and a xsl transformation

Davide Lissoni Mat.179878
17/11/2016

## 1.Introduction:

This assignment is about xml language and its aggregates languages used to manage it. In particular for this project has been used xml Schema and xslt technologies.
This report starts with a brief introduction on the mentioned above languages in order to explain in a better way how the project has been implemented.

Xml is a markup language that defines a set of rules for encoding documents in a format readable both for machine and human. XML is mostly used for the interchange of data over the Internet and basically it is structured in tags, elements and attributes.

XSD language (Xml schema) is used in order to create a formal description of a grammar for a markup language specifically based on XML. Xml schema allow us to specify the structure of our XML document and of each tag that can be used inside the xml. This is made using an approach based on Dtd adding the possibility of make a a greater control over the elements that may be inside a specific type of XML documents (i.e. the data type of a particular element or for example a predefined number of fields that an element can contains).

XSLT is the transformation language for xml, is based on xsl and has been created in order to transform xml document in other document types.

The requirements for this assignment have been divided in three main points:

1. Create an Xml Schema that best describe the ACMTrento.xml document;
2. Create an xsl transformation of the same xml document in order to create a csv document containing the following information:
   - Unit name;
   - Unit type;
   - Time.
3. Create a java program that, takes as input the xml and the xslt document created and then runs the transformation in the csv file required.

## 2.Implementation:

### *2.1 XML Schema:*

The xsd schema has been implemented in order to write the grammar of the xml document and in order to do that has been necessary to study the xml document.
The document xml contains a list of courses and their description, clustered in different areas.
The xml Schema implemented specify every data type of every element and attribute contained in the xml, if a attribute is required or not and some other grammar rules found in the document.
The remarkable rules for this schema are:

- The id of an area must be composed by two upper-case letters from A to Z

```
<xs:attribute  name="ID">
 <xs:simpleType>
   <xs:restriction base="xs:string">
     <xs:pattern value="[A-Z]{2}"/>
   </xs:restriction>
 </xs:simpleType>
</xs:attribute>
```

- The area must contains a sub-elements sequence;

```
<xs:complexType name="AREAType">
    <xs:sequence>
      <xs:element type="xs:string" name="AREA_NAME"/>
      <xs:element type="SHORT_NAMEType" name="SHORT_NAME"/>
      <xs:element type="xs:string" name="DESCRIPTION"/>
      <xs:element type="UNITSType" name="UNITS"/>
    </xs:sequence>
```

- The id of a unit must be composed by two upper-case letters from A to Z followed by numbers;

```
<xs:attribute name="ID" use="required">
<xs:simpleType>
 <xs:restriction base="xs:string">
        <xs:pattern value="[A-Z][A-Z]([0-9])+"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
```

- The type of a unit must be CORE or ELECTIVE;

```
<xs:attribute name="TYPE" use="required">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:pattern value="CORE|ELECTIVE"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
```

- The element UNIT must contain the sub elements UNIT_NAME and SHORT_NAME, TOPICS, LEARNING_OBJECTIVES and could contain the sub element TIME

```
<xs:complexType name="UNITType">
  <xs:choice maxOccurs="unbounded">
    <xs:element type="xs:string" name="UNIT_NAME" minOccurs="1"/>
    <xs:element type="SHORT_NAMEType" name="SHORT_NAME" minOccurs="1"/>
    <xs:element name="TIME" minOccurs="0">
```

## *2.2 XSLT File:*

The xslt has been implemented in order to satisfy the requirements given for this assignment and looks like as follow:

```
<xsl:stylesheet xmlns:xsl ="http://www.w3.org/1999/XSL/Transform"  version="1.0" >
<xsl:output method="text" omit-xml-declaration="yes" />
<xsl:template match="text()|@*"></xsl:template>
<xsl:template match="TAXONOMY/AREA/UNITS/UNIT">
"<xsl:value-of select="UNIT_NAME"/>"
<xsl:choose>
<xsl:when test="TIME = ''" >"time not present", </xsl:when>
<xsl:otherwise>"<xsl:value-of select="TIME"/>", </xsl:otherwise></xsl:choose>"<xsl:value-of select="@TYPE"/>"
<xsl:text>
</xsl:text>
</xsl:template>
</xsl:stylesheet>
```

The *<xsl:template match="text()|@*"></xsl:template>* line is used in order to take any kind of node present in the xml without do anything, in order to print only the nodes specified below this line. The same things could be managed using <xsl:for_each>.
The xslt then selects the value of the nodes to print out present in the TAXONOMY/AREA/UNITS/UNIT xml path.

## *2.3 Java Program:*

In order to run the xsl transformation has been implemented a simple java project:

```
public class Transformator {
    static Document document;
    public static void main(String[] argv) {
        if (argv.length != 2) {
            System.err.println("Usage: java Stylizer stylesheet xmlfile");
            System.exit(1);
        }

        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        try {
            File stylesheet = new File(argv[0]);
            File datafile = new File(argv[1]);

            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse(datafile);

            // Use a Transformer for output
            TransformerFactory tFactory = TransformerFactory.newInstance();
            StreamSource stylesource = new StreamSource(stylesheet);
            Transformer transformer = tFactory.newTransformer(stylesource);
            String outfile="output.csv";
            DOMSource source = new DOMSource(document);
            StreamResult result = new StreamResult(new FileOutputStream(outfile));
            transformer.transform(source, result);

        } catch (Exception tce) {

            System.out.println("\n** Transformation error");
            System.out.println("   " + tce.getMessage());

        }
    }
}
```

The transformation has been made through the transform() method of the abstract Transformer of TransformerFactory java class.

# 3.Deployment:

In order to compile the program make sure to be located in the project folder and then digit from command line:

*javac Transformator.java*

In order to run the application digit:

*java Transformator ACMTrento.xsl data/ACMTrento.xml*

If the system runs without error a file called output.csv will be created in the project folder. The csv file should looks like:

| | | |
|---|---|---|
| Functions, relations, and sets | 6 | CORE |
| | | |
| Basic logic | 10 | CORE |
| | | |
| Proof techniques | 12 | CORE |
| | | |
| Basics of counting | 5 | CORE |
| | | |
| Graphs and trees | 4 | CORE |