Java facilities in processing XML files – JAXB and generating PDF reports

Dănuţ-Octavian SIMION
Academy of Economic Studies, Bucharest, România
danut_so@yahoo.com

The paper presents the Java programming language facilities in working with XML files using JAXB (The Java Architecture for XML Binding) technology and generating PDF reports from XML files using Java objects. The XML file can be an existing one and could contain the data about an entity (Clients for example) or it might be the result of a SELECT-SQL statement. JAXB generates JAVA classes through xs rules and a Marshalling, Unmarshalling compiler. The PDF file is build from a XML file and uses XSL-FO formatting file and a Java ResultSet object.

Keywords: *Xml file, JAXB, Java classes, Java ResultSet object, Marshalling, Unmarshalling, XSL-FO formatting file.*

1 Introduction

In building Web applications or business application is very important to use different types of data structured in databases or in XML files. In our days many business corporations store their data in XML file files because it is easy to store and manipulate this type of files, offers many facilities in storing important data and can be used by many API's and programming languages and also they are available on different types of operating systems (Windows, Linux, Solaris, etc.). Java programming language is very oriented in working with this type of files and offers many facilities such as extracting data from XML files into Java classes or obtaining PDF reports from a Java ResultSet object based on a XML file which was the result of a SELECT-SQL statement [2], [3]. JAXB (*The Java Architecture for XML Binding*) is a technology that permits generating Java classes which corresponds with elements of an XML file [3], [4]. The reports represented in PDF files might be obtain from a XML file which is the result of a query SELECT-SQL, using Java technology and launching Apache FOP application for transforming the XML file into a PDF document.

2. Java facilities in processing XML files – JAXB

JAXB (*The Java Architecture for XML Binding*) is used for extracting data from an existing XML file into Java classes and from this point, the objects can be integrated in different types of applications [4], [5]. The principals facilities of JAXB are:

- ➢ Hides the necessary details of processing data;
- ➢ Offers a modality object oriented in working with XML files;
- ➢ Generating the Java classes is based on *xs* rules through a Marshalling, Unmarshalling compiler;
- ➢ Allows separating activities for programmers and web designers.

Here is an of a XML file that contain data about the entity *Clients.* This is the source code of this XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot>
 xmlns:od="urn:schemas-microsoft-
com:officedata"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="fis_clienti.xs
d" generated="2008-05-18T21:23:26">
<Clients>
<id_client>1</id_client>
<client_name>John Smith</ client_name >
<date_of_birth>1968-05-
18T00:00:00</date_of_birth>
<client_cnp>1723482374823</ client_cnp>
<client_adress>London, Cantebury
Street</client_adress>
</Clients>
<Clients>
<id_client>2</id_client>
<client_name>Arthur Seymour
</client_name>
<date_of_birth>1978-05-10T00:00:00</
date_of_birth >
```

```
<client_cnp>1343245346456</client_cnp>
<client_adress>Dublin, McDonald
 Street</client_adress>
</Clients>
</dataroot>
```

The principals steps for JAXB are:

2.1. Binding XML structure with Java classes.

This step contains the following activities:

- writing the *xs* document which contains the validations rules;
- creating/generating XML documents which follows the created schema;
- defining a schema for mapping XML unities and Java unities;
- generating Java classes from XML schema;
- compiling generated classes.

2.2. Binding XML datas with objects.

This step is realized through a client application which has a interface with the user and will contain the following activities:

- *Unmarshalling*: creating Java objects which corresponds with XML document, the event with validation;
- Modify the data from the objects tree through the corresponding application;
- *Marshalling*: saving the information back in the XML document and then validating this activity.

JAXB API – is an API for JAXB.

The JAXB technology is included in JWSDP (Java Web Services Developer Pack) with the following structure:

```
jwsdp
jaxb
bin <-- utilities for generating classes
(xjc)
docs <-- documentation
samples <-- applications examples
lib <-- jar archives for JAXB:
jaxb-api.jar, jaxb-xjc.jar, jaxb-impl.jar,
jaxb-libs.jaxp
lib <-- jaxp-api.jar
endorsed <-- archives for processing XML:
sax.jar, dom.jar, xercesImpl.jar, xalan.jar
jwsdp-shared
lib <-- jax-qname.jar, namespace.jar,
xslib.jar, relaxngDatatype
apache-ant
lib <-- ant.jar
```

2.2.1. Generating the classes

Generating the classes is made with the **xjc** – a schema compiler.

```
xjc [-options ...] <scheme_xs>
-nv: disable the validation of the document;
-b <file>: specifying a mapping file;
-p <package>: specifying the name of the
package which will be generated;
-classpath <file>: specify where are the user
classes;
-b <file>: specify a mapping file;
-xml schema: the file with rules is a XML
Schema file type (default);
-relaxng: the file file with rules is a RELAX
NG file type (not implemented);
-dtd: the file file with rules is a DTD file
type (not implemented);
```

For example here is the source code of *clients.xsd* file:

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="clients"
type="ClientsDef"/>
<xs:complexType name=" ClientsDef ">
<xs:sequence>
<xs:element name="pers" type="Person"
maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="Person">
<xs:sequence>
<xs:element name="name"
type="xs:string"/>
<xs:element name="date_of_birth"
type="xs:date"/>
<xs:element name="cnp"
type="xs:string"/>
<xs:element name="adress"
type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
```

Compilation of the schema

The compilation is made like this:

```
xjc catalog.xsd -p exemplu
```

and will have as a result creating a directory *example* which will contain:

Clients interface, ClientsDef interface, Person interface.

The directory *impl* which includes classe ClientsImpl, PersonImpl - specific classes JAXB

The generated Interfaces

```
package example;
public interface Person {
java.lang.String getName();
void setName(java.lang.String value);
java.sql.Date getDate_of_birth();
void setDate_of_birth (java.sql.Date value);
java.lang.String getCnp();
void setCnp(java.lang.String value);
java.lang.String getAdress();
void setAdress(java.lang.String value);
}
package example;
public interface ClientiDef {
java.util.List getPerson();
}
package example;
public interface Clients
extends javax.xml.bind.Element, exam-
ple.ClientsDef
{}
```

The generated classes:

Class PersonImpl corresponds for XML tag pers:

```
protected java.lang.String _Name;
protected java.sql.Date _Date_of_birth;
protected java.lang.String _Cnp;
protected java.lang.String _Adress;
java.lang.String getName() {...};
void setName(java.lang.String value) {...};
java.sql.Date getDate_of_birth() {...};
void  setDate_of_birth(java.sql.Date  value)
{...};
java.lang.String getCnp() {...};
void setCnp(java.lang.String value) {...};
java.lang.String getAdress() {...};
void setAdress(java.lang.String value) {...};
Class ClientsImpl coresponds XML tag clients:
protected  ListImpl  _Persoana  =  new  Lis-
tImpl(new java.util.ArrayList());
public java.util.List getPersoana() {...}
```

There is a default mapping between XSD data types and Java data types.

The Compilation of classes:

```
@echo off
set JAXB_LIBS=%JAXB_HOME%\lib
set JAXP_LIBS=%JWSDP_HOME%\jaxp\lib
set JWSDP_LIBS=%JWSDP_HOME%\jwsdp-shared\lib
set                   LIBS=%JAXB_LIBS%\jaxb-
api.jar;%JAXB_LIBS%\jaxb-ri.jar;
%JAXB_LIBS%\jaxb-xjc.jar;%JAXB_LIBS%\jaxb-
libs.jar;
%JAXP_LIBS%\jaxb-
api.jar;%JAXP_LIBS%\endorsed\xercesImpl.jar;
%JAXP_LIBS%\endorsed\xalan.jar;%JAXP_LIBS%\en
dorsed\sax.jar;
%JAXP_LIBS%\endorsed\dom.jar;
%JWSDP_LIBS%\jax-
qname.jar;%JWSDP_LIBS%\namespace.jar
javac -classpath %LIBS%;. exemplu\*.java ex-
emplu\impl\*.java
javac -classpath %LIBS%;. Main.java
java -classpath %LIBS%;. Main
```

2.2.2. Unmarshalling: from XML in Java.

For example here is the source code of a XML file named test.xml which correspond with the schema [3], [4], [5]:

```
<clients>
<person>
<name>John Smith</name>
<date_of_birth>1968-05-18T00:00:00</
date_of_birth >
<cnp_client>1723482374823</cnp_client>
<client_adress>London,              Cantebury
Street</client_adress >
</person>
<person>
<name>Arthur Seymour</name>
<date_of_birth>1978-05-
10T00:00:00</date_of_birth>
<cnp_client>1343245346456</cnp_client>
<adresa_client>Dublin,               McDonald
Street</adresa_client>
</person>
</clients>
```

Unmarshalling

```
import java.io.*;
import java.sql.*;
import java.util.*;
import javax.xml.bind.*;
import exemplu.*;
public class Main {
public static void main( String[] args ) {
try {
//1. Creating an JAXBContext object for mani-
pulating
// the utility classes from package example
JAXBContext       jc       =       JAXBCon-
text.newInstance("exemplu");
//2. Creating an Unmarshaller object
Unmarshaller u = jc.createUnmarshaller();
//3. Unmarshall: transfer the data from XML
in objects
Clients c = (Clients)u.unmarshal(
new FileInputStream("clients.xml"));
//4. The viewing of the information
viewing(c);
} catch(JAXBException e) {
e.printStackTrace();
} catch(IOException e) {
e.printStackTrace();
}
viewing (Clients c) {
for(Iterator  it  =  c.getPerson().iterator();
it.hasNext(); ) {
Person p = (Person)it.next();
System.out.println(  p.getName()  +  ":  "  +
p.getDate_of_birth() + ":" +
p.getCnp() + ":" + p.getAdress() + ".");
}}}
```

The validation of XML input data

This activity is made through the method setValidating(true) for the Unmarshaller object, before this action and handling the exceptions UnmarshallException which can be generated if XML datas are not like specified schemas [2], [3], [4].

```
try {
//2.1 Activating the validation
u.setValidating(true);
....
} catch(UnmarshalException e) {
System.out.println("Invalid Data: " + e);
}
```

2.2.3. Marshalling: from Java in XML

The actualization of data. Using the created classes can be accessed and modify the objects which represents XML information [4], [5].

```
//5. Processing/Updating information
ObjectFactory obj = new ObjectFactory();
Person p = obj.createPerson();
p.setNume("John Smith");
p.setDate_of_birth("18.05.1968");
p.setCnp("1723482374823");
p.setAdress("London, Cantebury Street");
List pers = c.getPerson();
pers.add(p);
afiseaza(c);
```

Marshalling

Transfer of the dates in XML format to an external destination represented by a stream of bytes.

```
//6. Marshall: transfer of the dates in XML
format
Marshaller m = jc.createMarshaller();
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPU
T, Boolean.TRUE);
m.marshal(c, System.out);
```

Validation of Java objects

The Java objects can be verified if its suited for XML transformation and if its corresponding with the specified schemas. This activity is made by an object named *Validator* :

```
try {
//5.1 Processing/Updating information
...
//5.2 Validation
Validator v = jc.createValidator();
boolean valid = v.validateRoot(c);
System.out.println(valid);
//6. Marshalling
...
} catch( ValidationException e) {
System.out.println( "Date invalide:" + e );
}
```

3. Java facilities in processing XML files – generating PDF reports.

The result represent a PDF file based on a XML file whom is the result of a query SELECT-SQL, using Java technology [1], [3], [5]. The steps for obtaining the PDF file are:

- ➢ defining a XML structure for the records of a data source;
- ➢ building a XSL-FO formatting file for representation of data in tabular form;
- ➢ elaborating the Java source for extracting records in a ResultSet object which will be iterate and will generate a XML file corresponding to the structure defined before this step;
- ➢ launching Apache FOP application for transforming the XML file into a PDF document.

3.1. Defining a XML structure for the records of a data source:

```
<Report>
<Title>Vizualizare Clienti</Title>
<Header>
<HeadData>Id Client</HeadData>
<HeadData> Client Name </HeadData>
<HeadData>Date_of_birth</HeadData>
<HeadData>Cnp Client</HeadData>
<HeadData>ClientAdress</HeadData>
</Header>
<Line>
<CellDataLeftAlign>id_client</ CellDataLeftAlign>
<CellDataCenterAlign> client_name</ CellDataCenterAlign>
<CellDataCenterAlign>date_of_birth</ CellDataCenterAlign>
<CellDataRightAlign>cnp_client</ CellDataRightAlign>
<CellDataLeftAlign>adress_client</ CellDataLeftAlign>
</Line>
</Report>
```

3.2. Building a XSL-FO formatting file for representation of data in tabular form:

The source code for building **formatPDF.xsl** file:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsl:stylesheet version="1.0"
xmlns:fo="http://www.w3.org/1999/XSL/Format"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" indent="yes"/>
<xsl:template match="Raport">
<fo:root
xmlns:fo="http://www.w3.org/1999/XSL/Format">
<fo: layout-master-set>
<fo:simple-page-master margin-right="1cm"
margin-left="1.5cm" margin-bottom="1cm" margin-top="1cm" page-width="29.7cm" page-height="21cm" master-name="PageMaster">
<fo:region-body margin-top="1cm"/>
<fo:region-before extent="1cm"/>
<fo:region-after extent="1cm"/>
</fo:simple-page-master>
</fo:layout-master-set>
<fo:page-sequence master-reference="PageMaster">
<!--header/footer-->
<fo:static-content flow-name="xsl-region-before">
fo:block text-align="start" font-size="10pt"
font-family="serif" line-height="1em + 4pt">
Page: <fo:page-number/>
</fo:block>
</fo:static-content>
<fo:static-content flow-name="xsl-region-after">
<fo:block text-align="start" font-size="10pt"
font-family="serif" line-height="1em + 4pt">
report test
</fo:block>
</fo:static-content>
<!--The contain of the page-->
<fo:flow flow-name="xsl-region-body">
<xsl:apply-templates select="Titlu"/>
<fo:table>
<fo:table-column column-width="15mm"/>
<fo:table-column column-width="40mm"/>
<fo:table-column column-width="20mm"/>
<fo:table-column column-width="25mm"/>
<fo:table-column column-width="80mm"/>
<fo:table-header>
<xsl:apply-templates select="Header"/>
<fo:table-header>
<fo:table-body>
<xsl:apply-templates select="Linie"/>
</fo:table-body>
</fo:table>
</fo:flow>
</fo:page-sequence>
</fo:root>
</xsl:template>
<!--Introducing the content-->
<xsl:template match="Title">
<fo:block font-size="16pt" font-family="sans-serif" font-weight="bold" space-after.optinum="15pt" text-align="center">
<xsl:apply-templates/>
</fo:block>
</xsl:template>
<xsl:template match="Header">
<fo:table-row>
<xsl:apply-templates/>
</fo:table-row>
</xsl:template>
<xsl:template match="HeadData">
<fo:table-cell font-size="12pt" font-weight="bold" border-style="solid" text-align="center">
<fo:block>
```

```
<xsl:apply-templates/>
</fo: block>
</fo:table-cell>
</xsl:template>
<xsl:template match="Line">
<fo:table-row>
<xsl:apply-templates/>
</fo:table-row>
</xsl:template>
<xsl:template match="CellDataLeftAlign">
<fo:table-cell   text-align="left"   font-
size="12pt"  font-family="Courier"  border-
style="none" padding-left="2pt">
<fo:block>
<xsl:apply-templates/>
</fo:block>
</fo:table-cell>
</xsl:template>
<xsl:template match="CellDataRightAlign">
<fo:table-cell   text-align="right"   font-
size="12pt"  font-family="Courier"  border-
style="none" padding-left="2pt">
<fo:block>
<xsl:apply-templates/>
</fo:block>
</fo:table-cell>
</xsl:template>
<xsl:template match="CellDataCenterAlign">
<fo:table-cell   text-align="center"   font-
size="12pt"  font-family="Courier"  border-
style="none" padding-left="2pt">
<fo:block>
<xsl:apply-templates/>
</fo:block>
</fo:table-cell>
</xsl:template>
</xsl:stylesheet>
```

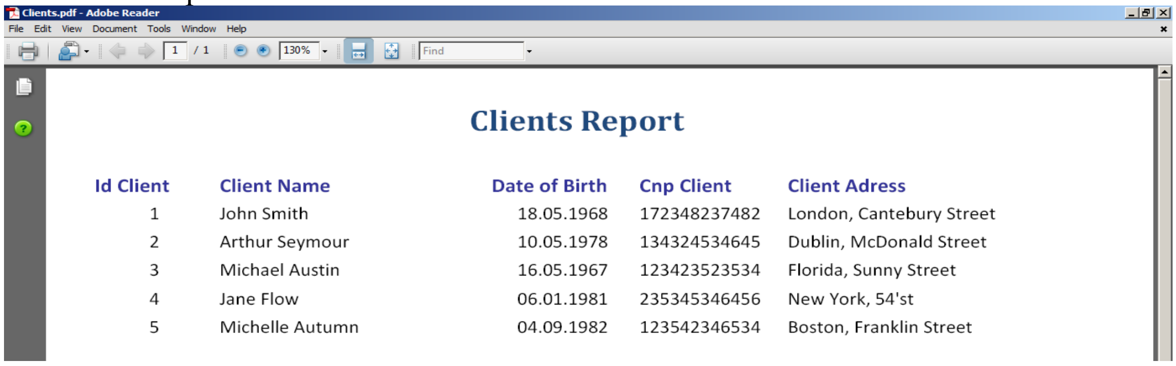
3.3. Elaborating the Java source for extracting records in a ResultSet object which will be iterate and will generate a XML file corresponding to the structure defined before this step:

The Java source code for the class which will generate the XML file with the corresponding structure used in formatting based on a **formatPDF.xsl** document. A SQL command will be executed and will be obtain a resulset that will be iterated and his lines will be written in a file on the disk (through the java.io package) [4], [5].

The Java source code:

```
package proiect_xml.reports.pdfreports;
import java.sql.*;
import java.io.*;
public class ReportClientsPDF {
static          String          directo-
ry_path="C:/PROJECT_XML/REPORTS/PDFREPORTS";
public static void main(String[] args) throws
Exception {
Connection
conn=project_xml.jdbc.Utility.getConexiune();
Statement stmt=conn.createStatement();
ResultSet      rs=stmt.executeQuery("select
id_client,    client_name,    date_of_birth,
cnp_client, client_adress from clients");
//initializing FileOutputStream
File  fisRaport=new  File(directory_path  +
"/reportData.xml");
FileOutputStream      fisStream=new      Fi-
leOut[utStream(fisRaport);
OutputStreamWriter   inscriptor=new   Output-
StreamWriter(fisStream);
BufferedWriter  bfInscriptor=new  BufferdWri-
ter(inscriptor);
// writing the report header
bfInscriptor.write("?xml  version=\"1.0\"  en-
coding=\"ISO-8859-2\"?>");         bfInscrip-
tor.newLine();
bfInscriptor.write("<Report>");    bfInscrip-
tor.newLine();
bfInscriptor.write("<Title>Clients        Re-
port</Title>"); bfInscriptor.newLine();
bfInscriptor.write("<Header>");    bfInscrip-
tor.newLine();
bfInscriptor.write("<HeadData>Id
Client</HeadData>");bfInscriptor.newLine();
bfInscriptor.write("<HeadData>Client
Name</HeadData>");bfInscriptor.newLine();
bfInscriptor.write("<HeadData>Date         of
Birth</HeadData>");bfInscriptor.newLine();
bfInscriptor.write("<HeadData>Cnp
Client</HeadData>");bfInscriptor.newLine();
bfInscriptor.write("<HeadData>Client
Adress</HeadData>");bfInscriptor.newLine();
//inchiderea nodului <Header>
bfInscriptor.write("</Header>");   bfInscrip-
tor.newLine();
// the text of the  report
while (rs.next())
{  bfInscriptor.write("<Line>");   bfInscrip-
tor.newLine();
bfInscrip-
tor.write("<CellDataLeftAlign>"+rs.getDouble(
"id_client")+"</CellDataLeftAlign>");
bfInscriptor.newLine();
bfInscrip-
tor.write("<CellDataCenterAlign>"+new
String(rs.getString("client_name").getBytes("
ISO-8859-1"),"ISO-
88592")+"</CellDataCenterAlign>"); bfInscrip-
tor.newLine();                     bfInscrip-
tor.write("<CellDataCenterAlign>"+new
String(rs.getDate("date_of_birth")
+"</CellDataCenterAlign>");        bfInscrip-
tor.newLine();
bfInscriptor.write("<CellDataRightAlign>"+new
String(rs.getString("cnp_client")
+"</CellDataRightAlign>");         bfInscrip-
tor.newLine();
bfInscriptor.write("<CellDataLeftAlign>"+new
String(rs.getString("client_adress")
+"</CellDataLeftAlign>");          bfInscrip-
tor.newLine();
bfInscriptor.write("</Line>");     bfInscrip-
tor.newLine();
}
// finalizing the report
bfInscriptor.write("</Report>);    bfInscrip-
tor.newLine();
bfInscriptor.close();
// Launching FOP
Org.apache.fop.apps.CommandLineOptions    op-
tions=null;
String[]                        params={"-
xml",directory_path+"/reportData.xml",
"-xsl",directory_path+"/formatPDF.xsl",
"-pdf",directory_path+"/report.pdf"};
options            =                    new
org.apache.fop.apps.CommandLineOptions(params
);
org.apache.fop.apps.Starter  starter  =  op-
tions.getStarter();
starter.run();
}}
```

The resulted Report in a PDF format:


Clients.pdf - Adobe Reader
File Edit View Document Tools Window Help
1 / 1
130%
Find

Clients Report

Id Client	Client Name	Date of Birth	Cnp Client	Client Adress
1	John Smith	18.05.1968	172348237482	London, Cantebury Street
2	Arthur Seymour	10.05.1978	134324534645	Dublin, McDonald Street
3	Michael Austin	16.05.1967	123423523534	Florida, Sunny Street
4	Jane Flow	06.01.1981	235345346456	New York, 54'st
5	Michelle Autumn	04.09.1982	123542346534	Boston, Franklin Street

Fig.1. Clients Report

4. Conclusions

In many applications the data sources are very different and a large variety of files must be included in the logic of those applications. The XML files are very common in most business corporations and they include semi structured data that might be used in different types of applications. The Java programming language offers many facilities in working with XML files such as extracting data from these files into Java classes and so they represents the source data for many applications or can obtain specific reports such as PDF documents [1], [2], [5]. In first case Java uses JAXB (The Java Architecture for XML Binding) technology that generates JAVA classes through xs rules and a Marshalling, Unmarshalling compiler. In second case the PDF document which is presented as a report is build from a XML file and uses XSL-FO formatting file and a Java ResultSet object. The XML file can be an existing one with an establish structure or can be the result of a query SELECT-SQL and using Java technology launching Apache FOP application for transforming the XML file into a PDF document is a easy way to obtain data in a standard report [4], [5].

References

[1] Doug Lea, *Java - Concurrent Programming in Java 2nd Edition,* Addison Wesley, 1999
[2] Earl Burden, *Java & Xml*, O'Reilly, 2001
[3] Brett McLaughlin, *Java And Xml Data Binding*, O'Reilly, 2002
[4] Elliotte Rusty Harold, *Processing XML with Java*, 2002
[5] Cowan, John and Richard Tobin, *XML Information Set*, World Wide Web Consortium, 2001

URI: http://xml.org/sax/properties/lexical-handler
URI: http://xml.org/sax/properties/declaration-handler
URI: http://xml.org/sax/properties/dom-node
URI: http://xml.org/sax/properties/xml-string
URI: http://java.sun.com