B.I.R.O.

Best Information through Regional Outcomes

A Public Health Project funded by the European Commission, DG-SANCO 2005

WORK PACKAGE 6

D6.1 DATABASE ENGINE

VERS. 0.1

WP LEADER: UNIVERSITY OF PERUGIA

Dept. of Internal Medicine
Via E. dal Pozzo
I-06126 Perugia Italy

Index

1. Work Package 6

1.1. Work Package overview

The Work Package 6 is led by University of Perugia (UNIPG). It started on 1st July 2006 and it will end on 31st July 2008.
The main referent of the WP6 is the Database Administrator at the Coordinating Centre.

1.2. General aim of Work Package 6

The aim of Work Package 6 is to develop the *BIRO Database Engine*.
The *BIRO Database Engine* will be used locally by each centre to connect to the local register, then to extract relevant data for BIRO purposes and finally to store them into a local BIRO database.

1.3. Specific aims of BIRO Database Engine

Specific purposes of *BIRO Database Engine* are:

- To be able to connect to any kind of DBMS
- To map centre's data to a BIRO compliant dataset
- To retrieve data from local database and store them into BIRO Export XML files following the specifications included in the BIRO Data Dictionary
- To create a BIRO local Database following the structure specified in the BIRO Data Dictionary
- To store data contained in XML files into the BIRO local database

1.4. Links to other Work Packages

BIRO Database Engine strictly relates to WP3 "Common Dataset" and WP4 "Data Dictionary" as they provided the definition of data that should be stored in the database, the XML schema for BIRO Export files and suggestions about the architecture of BIRO Database (relationships between different tables).
BIRO Database Engine will feed the WP8 Statistical Engine with all data needed to perform statistical analysis. *BIRO Database Engine* will also

interface with WP9 Communication Software whose aim is to develop solutions to securely transmit data from each centre to the Central Engine.
As *BIRO Database Engine* will manage sensible data, it will embed considerations from WP5 Privacy Impact Assessment.

1.5. Database Engine and BIRO Architecture

Figure 1 summarizes the role of WP6 Database Engine inside the overall BIRO Architecture.

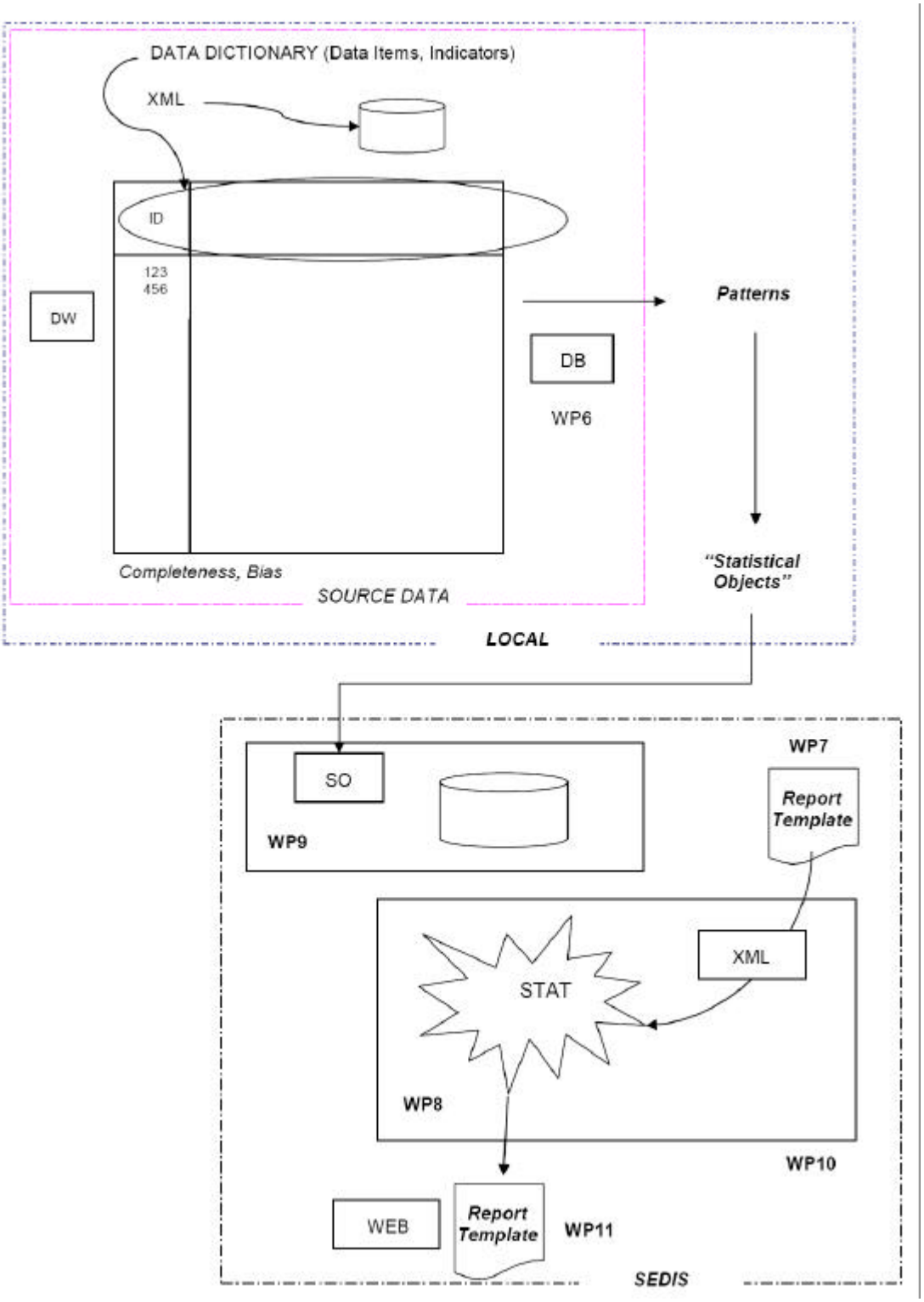


Figure 1: BIRO Architecture

The BIRO activity starts by formatting data the way the system expects it for further manipulation. Each partner should install the Database Engine into the local system. The Database Engine will produce "BIRO Export" files which

comply with specifications included in the Common Dataset and Data Dictionary (WP3,WP4). This will allow mapping centres' data to a BIRO compliant dataset. These files will be then loaded into the BIRO Database (WP6) that will manipulate data either directly, or via the statistical engine (WP8). The result will be statistical objects to be sent to the server via the communication software (WP9). As agreed during Privacy Impact Assessment (WP5), many privacy measures will be adopted in order to secure data sending: aggregation by group of patients, date fields approximated to time interval (e.g. months), pseudonym used for service centre, password access for local administrator prompting client program to send encrypted bundles to BIRO central server. The central server hosts the central engine (WP10). The central engine will be in charge of producing outputs as specified by the report templates (WP7). Both the central and statistical engines will require a programming language to "digest" instructions available in XML format to produce the final reports. Finally, all reports will be compiled to feed the web portal, as specified by the template protocol (WP11).

1.6. Major components of BIRO Database Engine

BIRO Database Engine tasks have been divided into two major sets and have been accomplished by mean of two separated software components. The interface between them is represented by the BIRO Export XML files. The first software component, which has been called *BIRO Adaptor*, connects to the local database and produces BIRO Export XML files. The second software component, which has been called *BIRO Database Manager*, reads the BIRO Export XML files and stores data into the local BIRO database.

2. BIRO Adaptor

The *BIRO Adaptor* is a small tool used to export data stored in a generic database to a XML file following the BIRO XML Schema.

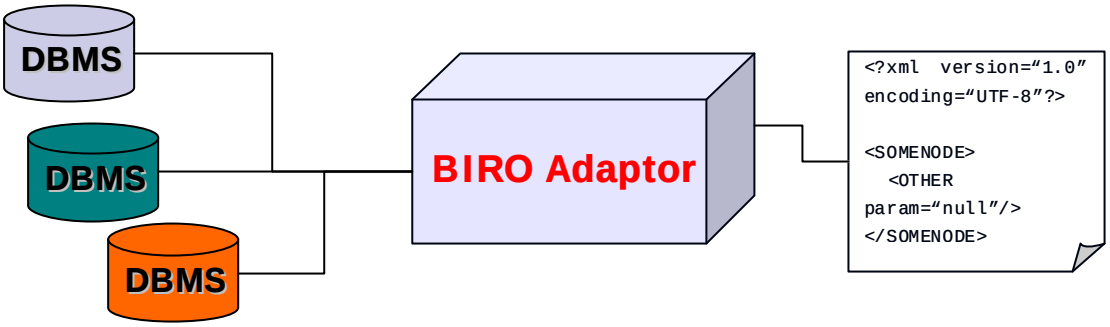


Figure 2: BIRO Adaptor schema

Written completely in Java language the tool connects to a DBMS using the JDBC standard layer, retrieves data using the SQL Query language, organizes data into BIRO structures, and writes down an XML file using the BIRO XML Schema.

This file will be sent to the database engine and imported into the BIRO central database.

Thanks to the JDBC layer this tool is able to connect and retrieve data from many different database engines provided we have the JDBC Java driver (available for many DBMS such as PostgreSQL, MySQL, SQL Server, Oracle, and so on…)

All the behaviour of this tool, from the database connection to what is to be exported, is controlled from a configuration file, read at startup.

In the next sections it is shown the entire configuration process, how to execute the adaptor, and an example of the exported XML file.

2.1. Configuration file

A simple Java standard property file is used to configure the adaptor. The configuration file is divided into different separated sections to simplify the process. Each section name, as in any standard *conf* file, is surrounded by **[ ]**.

A comment in the file starts with a # and will be ignored when getting properties.

Some options are mandatory and the adaptor does not work without them, others are optionally and are replaced by a default value if missing. In the following will be told if some option is either mandatory or optional.

2.1.1 JDBC section

The adaptor uses the JDBC layer to connect to any database system. For this to work we need to specify some JDBC parameters such as the JDBC driver class name to load, the database url (in the standard JDBC format), the user name and password.

While the driver, the database url and the user name are mandatory and have to be specified in the configuration file to correctly connect to a database, the password can be omitted in order to increase security and, should be the case, it will be asked during the execution.

Note that the specified driver class name must be specified as a fully qualified name and must be included in the class-path when starting the tool, otherwise the JDBC driver fails to load and a *ClassNotFoundException* will be thrown.

An example of this section in details is as follows:

```
# JDBC Connection properties, starting section
[JDBC]

# JDBC Driver to use, specify the full name of the Java class
Driver=org.postgresql.Driver
# JDBC url to connect to, specify the full jdbc url
Url=jdbc:postgresql:rrdm
# JDBC User name, specify the database username
Username=postgres
# JDBC Password, specify [optional] the password for the database
# If this field is empty it will be asked on input console
# for more security
Password=
```

2.1.2 Merge Table section

Once connected to the database the tool needs to know how data are stored in the database in order to be able to retrieve all the needed data. In this section it is specified the query to be executed in order to obtain a view where all patient entry are in rows with each data as a column. If the source database has already a view of this kind it is very simple to write down a query (simple SELECT * FROM “table”), otherwise there are two possible way of obtain this:

1. Preprocessing of data by creating a temporary table in the database before running the tool
2. Write a complex SQL query joining two or more tables in order to obtain a temporary VIEW

An example of this section in details is as follows:

```
# MergeTable information
[MergeTable]

# Specify the sql query to execute to retrieve all data
# to export in one table
SourceQuery=SELECT * FROM "MergeTable" ORDER BY "ID_CND", "EPIDATE"
;
# Specify the sql query to count rows to export
# (first column MUST be the one with the count)
CountQuery=SELECT COUNT(*) FROM (SELECT * FROM "MergeTable") AS N ;
# Specify date format of the date fields
DateFormat=dd/MM/yyyy
```

The query has to be specified in the mandatory SourceQuery parameter while the CountQuery (optional) is used to count exported data and to show a progress of the entire export process. It is simply a SELECT COUNT(*) FROM (<SourceQuery>) where <SourceQuery> is the query specified above.

If some export processes encounter problem due to parsing date problems (e.g. the country uses a different date format from the EU one) it is also possible to specify the date format found in the database in order to correctly export them.

2.1.3 BIRO XML section

Once retrieved data from the database the BIRO XML file has to be written. Since every database has a different table structure proved by the different SourceQuery parameter, we have to map every database column to a BIRO field so the *BIRO Adaptor* can successfully write the data correctly.

Some BIRO fields are not dependant on the datasource but they depends only on the country where data are stored, they are so called Static Fields in the sense that they do not change so these data can be specified directly in the configuration file. In the section details there's an example of the StaticFields from Perugia region

An example of this section in details is as follows:

```
# Static Fields
[SiteHeader]

DS_ID=2
DS_ADDRESS_1=Via E. Dal Pozzo
DS_ADDRESS_2=06126 Perugia, ITALY
DS_POST_CODE=06126
DS_COUNTRY=IT
DS_C_CONTACT=Massimo Massi Benedetti
DS_C_EMAIL=massi@unipg.it
DS_T_CONTACT=Pietro Palladino
DS_T_EMAIL=pietropalladino@gmail.com
```

The most fields, instead, are directly dependant on the column values of the retrieved data, so in this section it is specified, for each BIRO field, the corresponding column name in the retrieved query. If two or more columns, that in the database belong to different tables, have the same name you should rename them in the query, using the SQL *AS* operator, to avoid the ambiguity.

None of this values are mandatory but only the specified fields will be exported, so the more fields you have specified, the more complete the exported dataset is.

This section is divided into two sub-section. One describes the so called patient profile, data that are not subjected to change over time such as gender, date of birth, and so on… The other describes the Episode dataset, data that are taken on each patient visit.

An example of this section in details is as follows:

```
# Profile Fields
[Profile]

PAT_ID=id_CND
TYPE_DM=TIPODIAB
SEX=SESSO
DOB=NASCITA
DT_DIAG=DATADIAG

# Episode Fields
[Episode]

EPI_DATE=EPIDATE
WEIGHT=WEIGHT
HEIGHT=HEIGHT
```

2.2. Using the BIRO Adaptor

The adaptor is a batch process that does not need any user interaction. Once written the configuration file as the section 2.1 says, one can run the tool in a completely automatic way and sees after a while the exported XML file as output of the process.

The *BIRO Adaptor* is shipped in a single JAR file and can be executed as a standard Java process from the console with the following command:

```
java -Xmx1024m -cp BIROAdaptor.jar:<jdbcdriver>
   eu.biro.adaptor.BIROAdaptorMain <AdaptorConfig.conf>
   <Export>
```

where: (<..> are custom options)

- `java` is the Java Virtual Machine launcher
- `-Xmx1024m` is an option to allow the virtual machine to increase the maximum available memory since the data retrieval by the query is very memory consuming (increased to 1GB)
- `-cp` specify the class-path of the process
- `BIROAdaptor.jar` is the Adaptor jar file containing the main class
- `<jdbcdriver>` is the jar file containing the JDBC driver class, depends on the chosen DBMS
- `eu.biro.[…].BIROAdaptorMain` is the fully qualified name of the main class that starts the process
- `<AdaptorConfig.conf>` is the path of the configuration file to load
- `<Export>` is the export directory where the XML files will be stored

3. BIRO Database Manager

3.1. Aim of BIRO Database Manager

The aim of *BIRO Database Manager* is to parse BIRO Export XML files produced by *BIRO Adaptor* and to store data into the BIRO Database.

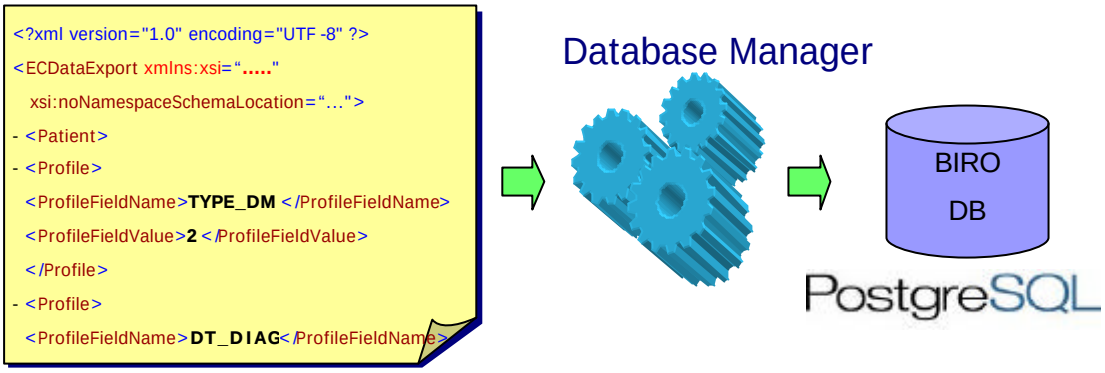


Figure 3: General aims of Database Manager

The data transfer from the XML document towards the database is performed by two steps, as shown in Figure 4. First XML files produced by *BIRO Adaptor* are parsed and data are translated in some appropriate Java Objects. This is called "unmarshalling" section as this term exactly means to create a mapping between elements of the XML document and members of a class to be represented in memory. The reverse process, to serialize a Java Object as XML, is called marshalling. The second step is the transformation of the hierarchically structured Java Objects into columns and records of the local BIRO database, which is simply called "storing" section.

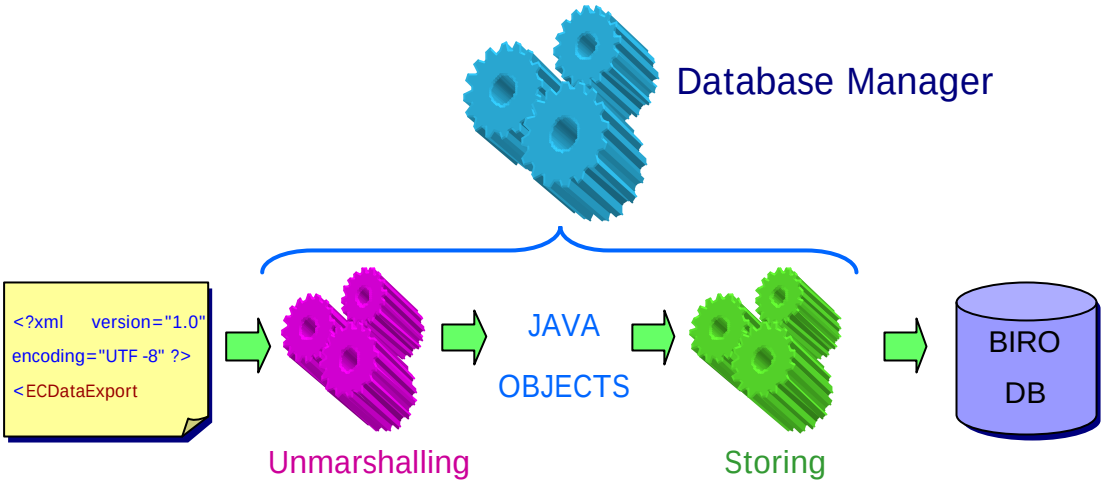


Figure 4: unmarshalling and storing sections

As described in Figure 5 the unmarshalling section in its turn is composed by two phases: first the unmarshaller parses BIRO XML Schema and creates an abstract Object Model. This model contains a java class for each element in the schema: Patient, Profile, EpisodeData, Data, etc.. Second the unmarshaller parses the BIRO Export XML files compliant with the BIRO XML Schema and produces instances of Patient, Profile, EpisodeData, Data, etc..
Also the storing section is composed by two phases: first the abstract Object Model is used to create the structure of the BIRO Database (tables, columns, primary keys, foreign keys, etc.). Then data contained in the fields of Java Objects are inserted as records into the appropriate tables of BIRO Database.

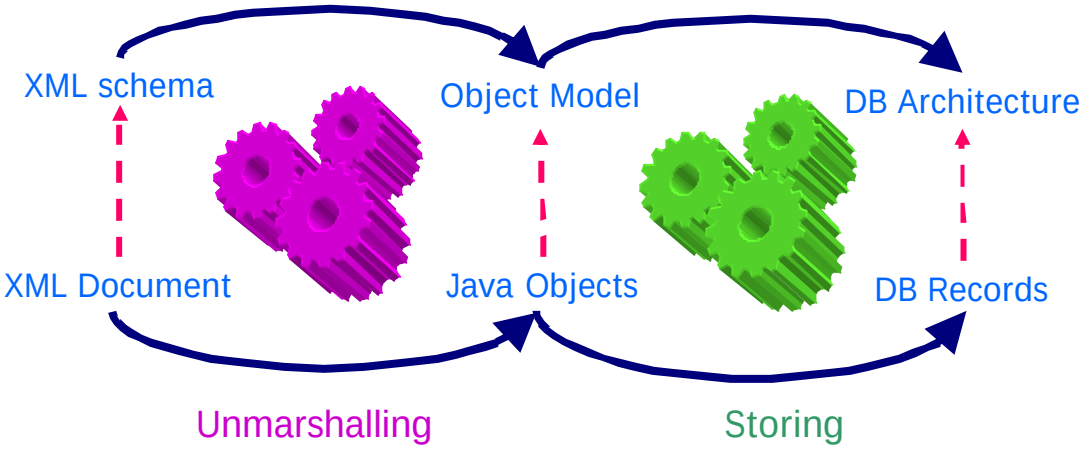


Figure 5: unmarshalling and storing sections in detail

3.2. Adopted tools

All the tools used to implement the *BIRO Database Manager* are free and open source. The DBMS chosen to host the BIRO Database is *PostgreSQL*. Unmarshalling process is accomplished by means of a data-binding framework called *Castor*, while storing section uses a persistence framework called *Hibernate*.

3.2.1 PostgreSQL

PostgreSQL is a powerful, open source relational database system. It has got many interesting features. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems. It is highly scalable both in the sheer quantity of data it can manage and in the number of concurrent users it can accommodate. *PostgreSQL* prides itself in standards compliance: its SQL implementation strongly conforms to the ANSI-SQL 92/99 standards. *PostgreSQL* is highly customizable, it runs stored procedures in more than a dozen programming languages. *PostgreSQL*'s source code is available under the most liberal open source license: the BSD license. This license gives you the freedom to use, modify and distribute *PostgreSQL* in any form you like, open or closed source. There are many high-quality GUI Tools available for *PostgreSQL* from both open source developers and commercial providers. *PostgreSQL* staff offers a strong and timely support to users. [6]

3.2.2 Castor

Castor is an Open Source data-binding framework for Java. It's the shortest path between Java objects, XML documents and relational tables. Three major features are *Castor XML*, *Source Code Generator* and *Castor JDO*.

Castor XML performs automatic XML data binding for class definitions which adhere to the Java Beans design pattern. Java to XML mapping automates transformation of Java objects to and from XML documents and provides Java object validation. It is possible to write XML based mapping file to specify XML bindings for existing object models.

The *Source Code Generator* can produce Java class definitions, XML binding information, and validation code based on a provided XML schema. The

Source Code Generator supports a "customization" binding file used in conjunction with an XML Schema for greater control over the generated source code.

Castor JDO automates Java persistence into any relational database server using JDBC. It uses an XML document to specify the mapping textually and provides an API to specify them programmatically. *Castor JDO* supports one-one and one-many relation types, SQL/Java inheritance, object graph to single row mapping, and multiple column primary keys and automatic type conversion. [1]

3.2.3 Hibernate

Hibernate is a powerful, high performance object-relational mapping (ORM) library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database. *Hibernate* is free as open source software that is distributed under the GNU Lesser General Public License.

Hibernate's primary feature is mapping from Java classes to database tables (and from Java data types to SQL data types). *Hibernate* also provides data query and retrieval facilities. *Hibernate* generates the SQL calls and relieves the developer from manual result set handling and object conversion, keeping the application portable to all SQL databases, with database portability delivered at very little performance overhead.

Hibernate provides transparent persistence for Plain Old Java Objects (POJOs). The only strict requirement for a persistent class is a no-argument constructor, not compulsorily public. *Hibernate* provides a dirty checking functionality. The latter feature allows to perform SQL update only on the modified fields of the persisted object in order to avoid unnecessary database write actions. *Hibernate* can be used both in standalone Java applications and in Java EE applications using servlets or EJB session beans. [3][4][5]

3.3. Technology overview

Using frameworks like *Castor* and *Hibernate* is not the only way to solve the problem of saving information of an XML documents into a database. Standard ways to access XML files make use *of W3C DOM API* and *SAX API*.

They allow the user to deal directly with the structural components (elements, attributes) of XML files, to manipulate XML at low level and extracting pieces of data. Whit frameworks instead, as they operate at an higher level of abstraction, all the XML manipulations are transparent to the user.

If we use standard API to persist XML files, we don't need to insert the intermediate step of Java Objects Model: we can extract strings from XML files and then execute queries to directly store them into the database.

The following table reports a comparison between the two approaches described above, highlighting the major drawbacks and advantages of each one. [7]

using standard API	using frameworks
↓ Write custom utilities to access the data is a laborious task, error-prone, and tricky to test and debug	↑ A very little effort is required to use frameworks, code is highly readable
↓ API deal with structure of XML file	↑ Creating Java Objects allow developers to deal with the logical model of data instead of their representation
↓ API parse the XML files and return simply strings which need to be converted into proper format to be manipulated	↑ Frameworks automatically convert string into desired format and also validate the data
↓ Code is very difficult to mantain and modify if changes to XML schema are necessary. Difficulties grow if XML schema is large.	↑ It is very easy to maintain and modify the DB engine using frameworks even if XML schema is large because all changes at low level are transparent to the user
↑ Working on XML files at low level allow the application to bind only the relevant data items, to avoiding the unnecessary processing that would be required to completely unmarshall	↓ Creation of Java Objects requires long time and much memory. Performance is worse.

the entire XML document and consequently to obtain high performances regarding both time and memory usage.	

3.4. Database Manager architecture

The *BIROClassGenerator* (see par. 5.1) allow one to read an XML schema and to automatically generate the source code of Java Classes relating to the elements in the XML schema. It is not included in the software package delivered to the partners but it is very useful for developers as it simplify the maintainability of *BIRO Database Manager* if changes to the XML schema are needed (see par. 3.7).

The *BIROClassGenerator* uses the *Castor* feature named *SourceGenerator.* It requires as arguments the file name of the BIRO XML schema, the output directory where Java Classes will be stored and the file name of the *binding file.* The *binding file* (see par. 5.2) is an XML file used to specify to the *SourceGenerator* some constraints which have to be respected while generating Java source code. The default binding used to generate the Java Object may not meet your expectations. For instance, the default binding doesn't deal with naming collisions that can appear because XML Schema allows an element declaration and a complexType definition to use the same name. The source generator will attempt to create two Java classes with the same qualified name. However, the second class generated will simply overwrite the first one. User may also need to change the default datatype binding provided by *Castor* or to implement his own validation rules. [2]

In our case the binding file has been used to specify the datatype *java.util.ArrayList* instead of the default *Java.util.Vector* for collections.

The generated Java Classes are represented with an UML diagram in the par. 5.3 .

The mapping between Java Object Model and Relational Model is specified by means of one or more XML documents named mapping files. The mapping file is designed to be Java centric: for each class and each attribute, the

document specifies the associated table and column and constraints. The mapping files written for the *BIRO Database Manager* application are reported at par. 5.4. These files are used during the storing section to create the database architecture and to store at the right place the unmarshalled Java Objects.

The database structure is represented at par. 5.5: the complete SQL code and the UML diagram of the database are included.

BIRODatabaseManagerMain (see par. 5.6) is the main class used to start the *Database Manager*. It requires as input the file name of the XML Configuration file and the input directory containing the BIRO Export XML files. Each file in this directory is sent to the *UnmarshallingAndStoringManager* (see par. 5.7) which calls the appropriate *Castor* utilities to unmarshall the file and then calls the appropriate *Hibernate* utilities to store the Java Objects into the database.

Some utilities have also been implemented: *HibernateUtil* (see par. 5.8.1) allow *Hibernate* functions to read the Configuration File where database connection settings are stored; *Loader* (see par. 5.8.2) is an utility class for Jar resources.

3.5. Configuration file

A simple Java standard property file is used to configure the *BIRO Database Manager*. Properties are mandatory and the *BIRO Database Manager* does not work without them. A comment in the file starts with a # and will be ignored when getting properties. As *BIRO Database Manager* may be used with any DBMS, some properties referring to JDBC parameters have to be specified: the JDBC driver class name to load, the *Hibernate* dialect which allow Hibernate to generate SQL optimized for a particular relational database, the database url (in the standard JDBC format), the user name and password.

An example of the configuration file in details is as follows:

```
#This is the BIRO Database Manager Configuration File

#the following lines specify the JDBC properties

#JDBC Driver to use, specify the full name of the Java class
hibernate.connection.driver_class = org.postgresql.Driver
```

```
#Hibernate Dialect
hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect

#JDBC url to connect to, specify the full jdbc url
hibernate.connection.url = jdbc:postgresql://localhost/BIROdatabase

# JDBC User name, specify the database username
hibernate.connection.username = myuser

# JDBC Password for the database
hibernate.connection.password = secret
```

3.6. Using the Database Manager

The *BIRO Database Manager* is a batch process that does not need any user interaction. Once installed the DBMS and written the configuration file as the section 3.5 says, one can run the tool in a completely automatic way and sees after a while the BIRO database populated with data of BIRO export XML files as output of the process.

The *BIRO Database Manager* is shipped in a single JAR file and can be executed as a standard Java process from the console with the following command:

```
java -Xmx1024m -cp BIRODatabaseManager.jar:<jdbcdriver>
   test.BIRODatabaseManagerMain
   <BIRODatabaseManagerConfig.conf> <Import>
```

where: (<..> are custom options)

- `java` is the Java Virtual Machine launcher
- `-Xmx1024m` is an option to allow the virtual machine to increase the maximum available memory since the data retrieval by the query is very memory consuming (increased to 1GB)
- `-cp` specify the class-path of the process
- `BIRODatabaseManager.jar` is the *BIRO Database Manager* jar file containing the main class
- `<jdbcdriver>` is the jar file containing the JDBC driver class, depends on the chosen DBMS
- `test.BIRODatabaseManagerMain` is the fully qualified name of the main class that starts the process
- `<BIRODatabaseManagerConfig.conf>` is the path of the configuration file to load

- `<Import>` is the export directory where the BIRO Export XML files have been stored

3.7. Database Manager maintainability

The *BIRO Database Manager* is very easy to maintain if changes occur to BIRO XML schema. It is sufficient to run the *BIROClassGenerator* to automatically get the new Java Object Model. Then, the developer has only got to update the XML mapping files which link the classes of the Java Object Model to the tables of the BIRO database. Finally, the *BIRODatabaseManager* automatically changes the database structure according to the new mapping file and stores data again.

3.8. Database Manager performance

Creating Java Object requires a long time and much memory. In order to improve *BIRO Database Manager* performance we chose to import data of each patient separately which means that *BIRO Adaptor* is configured to produce many little *BIRO XML Export* files instead of a big one for all patients. Moreover we forced *Castor* to use a different parsing tool named *Piccolo* which is faster than the native one (*Xerces*) and we switched off the logging of SQL statement to console.

4. BIRO Adaptor Source Code

4.1. Main Class

The main Class used to start the BIRO Adaptor

```
/**
 * Project: BIRO-Project (Funded by European Commission 2005-2008)
 *
 * File:     BIROAdaptorMain.java
 * Author:   Pietro Palladino
 *
 * License: TO Be Written. - Free and Open to All BIRO Partners
 *
 */
package eu.biro.adaptor;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.sql.SQLException;

import org.plibrary.util.PropertyExt;

/**
 * This simple class allow one partner to read the configuration file (see
documentation),
 * connect to source database and export an XML File following BIRO Schema.
<br>
 *
 * @author Pietro Palladino
 */
public final class BIROAdaptorMain {

    private BIROAdaptorMain() {
        throw new Error("Cannot instantiate Main");
    }

    /**
     * Execute BIRO Adaptor<br>
     * It requires as arguments:<br>
     * 0) Input File Name of the XML Configuration<br>
     * 1) Output Directory of the BIRO Xml files. Password is read from the
XML
     * File, if it is not defined it will be asked from the standard input
     */
    public static void main(String[] args) {
        if (args.length < 2) {
            System.err.println("\nUsage:\n\t" +
BIROAdaptorMain.class.getName()
                    + " [ConfigFile] [OutputDir]");
            return;
        }

        try {
            String conf = args[0];
            File in = new File(conf);
            if (!in.exists())
                throw new FileNotFoundException("Cannot find the
Configuration file at '"
                        + conf + "'");
            String dir = args[1];

            PropertyExt ext = new PropertyExt(in);
            BIROAdaptor adaptor = new BIROAdaptor(ext);
            adaptor.createBIROFile(dir);
        }
        catch (ClassNotFoundException e) {
            System.out.println("Something wrong with JDBC Driver for:\n"
                    + e.toString());
        }
```

```
        catch (SQLException e) {
            System.out.println("Something wrong with SQL Access for:\n"
                    + e.toString());
        }
        catch (IOException e) {
            System.out.println("Something wrong with IO Access for:\n"
                    + e.toString());
        }
    }
}
```

4.2. The BIRO Adaptor

The class that connects to database, retrieves data and creates XML files

```
/**
 * Project: BIRO-Project (Funded by European Commission 2005-2008)
 *
 * File:      BIROAdaptor.java
 * Author:    Pietro Palladino
 *
 * License: TO Be Written. - Free and Open to All BIRO Partners
 *
 */
package eu.biro.adaptor;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Map.Entry;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;

import org.plibrary.util.CharList;
import org.plibrary.util.ProgressCalculator;
import org.plibrary.util.PropertyExt;

/**
 * This simple class allow one partner to read the configuration file (see
 * documentation), connect to source database and export an XML File
following
 * BIRO Schema.
 *
 * @author Pietro Palladino
 */
public final class BIROAdaptor {

    private final Date today = new Date();
    private final PropertyExt properties;
    private final CharList document;
    private final SimpleDateFormat writer;

    private int currentID = 1;
    private BufferedWriter out;
    private ZipOutputStream zout;
    private ProgressCalculator calc;
    private Connection conn;
    private ResultSet set;
    private SimpleDateFormat reader;

    /**
     * Creates new BIROAdaptor
     *
```

```
     * @param pExt
     *            Property file for configuring Adaptor
     * @see PropertyExt
     */
    public BIROAdaptor(PropertyExt pExt) {
        this.properties = pExt;
        this.document = new CharList();
        this.writer = new SimpleDateFormat("yyyy-MM-dd");
    }

    /**
     * Creates the Export BIRO File
     *
     * @param dir
     *            the name of the directory
     * @throws ClassNotFoundException
     *             if jdbc driver cannot be loaded
     * @throws SQLException
     *             if something wrong happened in the db connection
     * @throws IOException
     *             if something wrong happened in the IO stream
     */
    public void createBIROFile(String dir) throws ClassNotFoundException,
SQLException, IOException {
        try {
            conn = initConnection();
            int c = getNumberOfRows();
            set = readData(c);
            startDirectory(dir);
            writeSourceExport();
            writeData();
            calc.done();
            calc.printGraphicOutput();
            System.out.println("BIRO Export file created.");
        } finally {
            endFile();
            if (conn != null)
                conn.close();
            System.out.println();
        }
    }

    private Connection initConnection() throws ClassNotFoundException,
SQLException {
        properties.changeSection("JDBC");
        String driver = properties.getProperty("Driver");
        String url = properties.getProperty("Url");
        String user = properties.getProperty("Username");
        String pwd = properties.getProperty("Password");
        while (pwd == null) {
            System.out.println("Insert Database password: [ECHO disabled]");
            pwd = new String(System.console().readPassword());
        }
        Class.forName(driver);
        return DriverManager.getConnection(url, user, pwd);
    }

    private int getNumberOfRows() throws SQLException {
        ResultSet r = null;
        try {
            properties.changeSection("MergeTable");
            String query = properties.getProperty("CountQuery");
            Statement st = conn.createStatement();
            System.out.print("Reading number of row to export .. ");
            r = st.executeQuery(query);
            int c = r.next() ? r.getInt(1) : 0;
            if (c == 0)
                throw new RuntimeException("Merge table is empty!");
            System.out.println(c + " rows.");
            calc = new ProgressCalculator("BIRO XML", c);
            return c;
        } finally {
            if (r != null)
                r.close();
        }
    }
```

```
    private ResultSet readData(int c) throws SQLException {
        calc.startPrintRotors("Reading " + c + " values [may take a
while]");
        properties.changeSection("MergeTable");
        String query = properties.getProperty("SourceQuery");
        reader = new SimpleDateFormat(properties.getProperty("DateFormat"));
        Statement st = conn.createStatement();
        ResultSet r = st.executeQuery(query);
        calc.stopPrintRotors();
        System.out.println();
        calc.printGraphicOutput();
        return r;
    }

    private void startDirectory(String dirName) throws IOException {
        properties.changeSection("General");
        boolean zip = Boolean.valueOf(properties.getProperty("ZIP"));

        File dir = new File(dirName);
        dir.mkdirs();

        if (zip) {
            final String fileName = "Export.zip";

            File f = new File(dir, fileName);

            if (f.exists()) {
                System.out.println(" - WARNING - : Attempting to overwrite
file " + f.getName() + " ..        ");

                if (!f.canWrite())
                    throw new IOException("Export file already exists and
cannot be overwritten!");
            }
            FileOutputStream fo = new FileOutputStream(f);
            zout = new ZipOutputStream(fo);
            zout.setLevel(9);
            zout.setMethod(ZipOutputStream.DEFLATED);
            zout.setComment("BIRO Export File created on " +
getStringDate(today) + " -->");
        }
    }

    private void startEntry(String name) throws IOException {
        flush();
        OutputStream o;
        if (zout == null)
            o = new FileOutputStream(name);
        else {
            ZipEntry e = new ZipEntry(name);
            zout.putNextEntry(e);
            o = zout;
        }
        out = new BufferedWriter(new OutputStreamWriter(o));
        writeHeader();
    }

    private void writeSourceExport() throws IOException {
        startEntry("DataSource.xml");
        beginFileWithNode("ECDataSourceExport");
        writeSiteHeader();
        writeSiteProfile();
        writeExportProfile();
        endnode("ECDataSourceExport");
    }

    private void writeHeader() {
        document.append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
        document.newLine();
        document.append("<!-- BIRO Export File created on " +
getStringDate(today) + " -->");
    }

    /**
     * Writes Site Header. Can be overridden to change static data
```

```
     */
    private void writeSiteHeader() {
        startnode("SiteHeader");
        node("DateHeaderInformationChecked", getStringDate(today));
        for (Entry<String, String> e :
properties.getPropertiesInSection("SiteHeader"))
            node(e.getKey(), e.getValue());
        endnode("SiteHeader");
    }

    /**
     * Writes Site Profile. Can be overridden to change static data
     */
    private void writeSiteProfile() {
        startnode("SiteProfile");
        node("DateProfileInformationChecked", getStringDate(today));
        for (Entry<String, String> e :
properties.getPropertiesInSection("SiteProfile"))
            node(e.getKey(), e.getValue());
        endnode("SiteProfile");
    }

    private void writeExportProfile() {
        for (String section : new String[] { "Profile", "Episode" }) {
            for (Entry<String, String> e :
properties.getPropertiesInSection(section)) {
                String biro = e.getKey();
                String source = e.getValue();
                if (isNull(source))
                    writeNotPresent(biro);
                startnode("FieldExportProfiles");
                node("FieldName", biro);
                node("DateStatusLastReviewed", getStringDate(today));
                node("Mandatory",
biro.equals(properties.getPropertyInSection("BIROFields", "PatientID")));
                node("Recorded", !isNull(source));
                node("Consistency", isNull(source) ? "Low" : "High");
                node("Completeness", isNull(source) ? "0" : "100");
                node("Routine", !isNull(source));
                node("QualityScore", isNull(source) ? "Low" : "High");
                endnode("FieldExportProfiles");
            }
        }
    }

    private void switchPatient() throws IOException {
        startEntry((currentID++) + ".xml");
        beginFileWithNode("ECDataExport");
        startnode("Patient");
    }

    private void writeData() throws IOException, SQLException {
        try {
            Date lastDate = null;
            String lastId = null;
            boolean samePat = false;
            String epidate = properties.getDerivedProperty("EpisodeDate");
            String pid = properties.getDerivedProperty("PatientID");
            while (set.next()) {
                String id = set.getString(pid);
                if (isNull(id))
                    continue;
                if (!id.equals(lastId)) {
                    if (lastDate != null) {
                        endnode("EpisodeData");
                        lastDate = null;
                    }
                    if (lastId != null) {
                        endnode("Patient");
                        endnode("ECDataExport");
                    }
                    lastId = id;
                    switchPatient();
                    samePat = false;
                } else
                    samePat = true;
```

```
                if (!samePat) {
                    for (Entry<String, String> e :
properties.getPropertiesInSection("Profile")) {
                        String v = set.getString(e.getValue());
                        if (isNull(v))
                            continue;
                        startnode("Profile");
                        node("ProfileFieldName", e.getKey());
                        node("ProfileFieldValue", v);
                        endnode("Profile");
                    }
                }

                String s_now = set.getString(epidate);
                Date now = null;
                try {
                    if (s_now != null)
                        now = reader.parse(s_now);
                } catch (ParseException e) {
                    System.out.println(" - WARNING - : " + e.toString() + "
with pattern '" + reader.toPattern() + "' ..");
                }
                if (now != null) {
                    if (!now.equals(lastDate)) {
                        if (lastDate != null)
                            endnode("EpisodeData");
                        lastDate = now;
                        startnode("EpisodeData");
                        node("EpisodeDate", getStringDate(now));
                    }
                    for (Entry<String, String> e :
properties.getPropertiesInSection("Episode")) {
                        String v = set.getString(e.getValue());
                        if (isNull(v))
                            continue;
                        startnode("Data");
                        node("EpisodeFieldName", e.getKey());
                        node("EpisodeFieldValue", v);
                        endnode("Data");
                    }
                }
                calc.doStep();
                if (calc.hasChanged()) {
                    flush();
                    calc.printGraphicOutput();
                }
            }
            if (lastId != null)
                endnode("Patient");
        } finally {
            if (set != null)
                set.close();
            endnode("ECDataExport");
        }
    }

    private String getStringDate(Date d) {
        return writer.format(d);
    }

    private boolean isNull(Object v) {
        return "null".equals(String.valueOf(v));
    }

    protected void writeNotPresent(String field) {
        document.newLine();
        document.append("<!-- BIRO Field '" + field + "' has no value in
DataBase -->");
    }

    private void endFile() throws IOException {
        flush();
        if (zout != null) {
            zout.closeEntry();
            zout.flush();
            zout.close();
```

```
        }
    }

    private void beginFileWithNode(String node) {
        document.newLine();
        document.append("<").append(node);
        document.append(" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-
instance\""
                + " xsi:noNamespaceSchemaLocation=\"../" + node +
".xsd\">");
        document.addIndentation(false);
    }

    private void startnode(String node) {
        document.newLine();
        document.append("<").append(node).append(">");
        document.addIndentation(false);
    }

    private void endnode(String node) {
        document.removeIndentation();
        document.append("</").append(node).append(">");
    }

    protected void node(String node, Object value) {
        document.newLine();
        document.append("<").append(node).append(">");
        document.append(isNull(value) ? "null" : value);
        document.append("</").append(node).append(">");
    }

    private void flush() throws IOException {
        if (out == null)
            return;
        if (document != null)
            document.newLine();
        out.write(document.toString());
        out.flush();
        document.clear();
    }
}
```

4.3. Common utilities

Some utilities that simplify the *BIRO Adaptor* work. All classes are taken from PLibrary version 1.2.6 and are freely released to BIRO consortium by the author. These classes are not supposed to be used outside BIRO Project.

4.3.1 CharList

A high performance replacement for StringBuilder allowing automatic indentation

```
/**
 * Project: BIRO-Project (Funded by European Commission 2005-2008)
 *
 * File:     CharList.java
 * Author:   Pietro Palladino
 *
 * License: TO Be Written. - Free and Open to All BIRO Partners
 *
 */
package org.plibrary.util;

import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.Serializable;
import java.nio.CharBuffer;
import java.util.Iterator;
```

```
/**
 * A String creator, high performances (than StringBuilder & comp.) for
 * inserting and deleting characters in any positions, especially when are
ones
 * next to the others
 *
 * @author Pietro
 * @version 1.2.2
 */
public final class CharList
        extends Object
        implements Iterable<CharNode>, Serializable, CharSequence {

    private static final long serialVersionUID = 7414670957881159883L;

    protected final CharNode head;
    private CharNode current;
    private int pos = 0;
    private int size = 0;
    private int indentation = 0;
    private String tab = "  ";

    /**
     * Creates a new CharList
     */
    public CharList() {
        head = new CharNode('\0');
        current = head;
        pos++;
    }

    /**
     * Creates a new CharList appending this string
     *
     * @param s
     *            the new string
     */
    public CharList(String s) {
        this();
        insert(1, s);
    }

    /**
     * Creates a new CharList appending these strings
     *
     * @param prefix
     *            the prefix for each string
     * @param postfix
     *            the postfix for each string
     * @param ss
     *            the new strings
     */
    public CharList(String prefix, String postfix, String... ss) {
        this();
        for (String s : ss)
            append(prefix).append(s).append(postfix);
    }

    /**
     * Creates a new CharList appending these objects
     *
     * @param prefix
     *            the prefix for each string
     * @param postfix
     *            the postfix for each string
     * @param <O>
     *            the type of Object
     * @param oo
     *            the new objects
     */
    public <O extends Object> CharList(String prefix, String postfix, O...
oo) {
        this();
        for (O o : oo)
            append(prefix).append(o).append(postfix);
```

```
    }

    /**
     * Creates a new CharList appending this character
     *
     * @param c
     *            the new character
     */
    public CharList(char c) {
        this();
        insert(1, c);
    }

    /**
     * Inserts this character at specified index
     *
     * @param index
     *            from 0 to size
     * @param c
     *            the new character
     * @return this
     */
    public CharList insert(int index, char c) {
        find0(index);
        insert(new CharNode(c));
        return this;
    }

    /**
     * Inserts this string at specified index
     *
     * @param index
     *            from 0 to size
     * @param s
     *            the new string
     * @return this
     */
    public CharList insert(int index, String s) {
        if (s == null)
            return this;
        int i = index;
        for (char c : s.toCharArray())
            insert(i++, c);
        return this;
    }

    /**
     * Appends this character at the end
     *
     * @param c
     *            the new character
     * @return this
     */
    public CharList append(char c) {
        insert(size, c);
        return this;
    }

    /**
     * Appends this string at the end
     *
     * @param s
     *            the new string
     * @return this
     */
    public CharList append(String s) {
        insert(size, s);
        return this;
    }

    /**
     * Sets indentation on new line +1
     *
     * @deprecated this method adds always a new line
     * @see #addIndentation(boolean)
     * @since 1.2.2
```

```
     */
    @Deprecated
    public CharList addIndentation() {
        this.indentation++;
        return newLine();
    }

    /**
     * Sets indentation on new line +1
     *
     * @param newLine
     *            true if should create new line
     * @since 1.2.6
     */
    public CharList addIndentation(boolean newLine) {
        this.indentation++;
        return newLine ? newLine() : this;
    }

    /**
     * Sets indentation on new line -1
     *
     * @since 1.2.2
     */
    public CharList removeIndentation() {
        this.indentation--;
        return newLine();
    }

    /**
     * Sets indentation on new line = 0
     *
     * @since 1.2.2
     */
    public CharList restoreIndentation() {
        this.indentation = 0;
        return newLine();
    }

    /**
     * Set indentation size
     *
     * @param size
     */
    public void setIndentationSize(int size) {
        char[] tab = new char[size];
        for (int i = 0; i < size; i++)
            tab[i] = ' ';
        this.tab = new String(tab);
    }

    /**
     * Returns length of indentation tab
     *
     * @return tab size
     * @since 1.2.2
     */
    public int getIndentationSize() {
        return tab.length();
    }

    /**
     * Adds a new line
     *
     * @since 1.2.2
     */
    public CharList newLine() {
        append(System.getProperty("line.separator"));
        for (int i = 0; i < indentation; i++)
            append(tab);
        return this;
    }

    /**
     * Appends o.toString() at the end
     *
```

```
     * @param <O>
     *            the type
     * @param o
     *            the object
     * @return this
     */
    public <O> CharList append(O o) {
        return append(o.toString());
    }

    /**
     * Deletes all characters between start and end-1
     *
     * @param start
     *            the start
     * @param end
     *            the end
     * @return this
     */
    public CharList delete(int start, int end) {
        find0(start);
        CharNode n = current;
        for (int i = start; i < end; i++)
            n = n.getNext();
        current.setNext(n.getNext());
        size -= end - start;
        return this;
    }

    /**
     * Deletes the last character
     */
    public void deleteLast() {
        delete(size() - 1, size());
    }

    /**
     * Clear list
     */
    public void clear() {
        delete(0, size());
    }

    /**
     * Returns a substring
     *
     * @param start
     *            start index
     * @param end
     *            end index
     * @return this
     */
    public String substring(int start, int end) {
        find0(start);
        CharBuffer b = CharBuffer.allocate(end - start + 1);
        CharNode n = current;
        for (int i = start; i <= end; i++) {
            b.append(n.getChar());
            n = n.getNext();
        }
        b.rewind();
        return b.toString();
    }

    /**
     * Returns first index of String
     */
    public int indexOf(String str) {
        return indexOf(str, 0);
    }

    public int indexOf(String str, int fromIndex) {
        int i = 0;
        int j = -1;
        int l = str.length();
        for (CharNode n : this) {
```

```
            if (++j < fromIndex)
                continue;
            if (n.getChar() == str.charAt(i))
                i++;
            else
                i = 0;
            if (i == l)
                return j - 1 + 1;
        }
        return -1;
    }

    /**
     * Useful for 'foreach cycles' Used by toString()
     *
     * @return an iterator
     */
    public Iterator<CharNode> iterator() {
        return new Iterator<CharNode>() {

            protected CharNode now = head.getNext();
            protected boolean started;

            /**
             * @see java.util.Iterator#hasNext()
             */
            public boolean hasNext() {
                return started ? now.getNext() != null : now != null;
            }

            /**
             * @see java.util.Iterator#next()
             */
            public CharNode next() {
                if (!started) {
                    started = true;
                    return now;
                }
                return now = now.getNext();
            }

            /**
             * @see java.util.Iterator#remove()
             */
            public void remove() {
                throw new UnsupportedOperationException("This Iterator does
not support remove");
            }
        };
    }

    /**
     * Returns the String, high performance with java.nio
     *
     * @return the string
     */
    @Override
    public String toString() {
        CharBuffer b = CharBuffer.allocate(size);
        for (CharNode n : this)
            b.append(n.getChar());
        b.rewind();
        return b.toString();
    }

    public void printString(OutputStream o) {
        PrintWriter w = new PrintWriter(new OutputStreamWriter(o), true);
        w.write(toString());
        w.flush();
    }

    public void printStringToOut() {
        printString(System.out);
    }

    private void insert(CharNode node) {
```

```
        try {
            node.setNext(current.getNext());
        }
        catch (Exception e) { // This is ignored
        }
        current.setNext(node);
        current = node;
        pos++;
        size++;
    }

    public int size() {
        return size;
    }

    /**
     * Returns the length of this character sequence. The length is the
number
     * of 16-bit chars in the sequence.
     *
     * @return the number of chars in this sequence
     */
    public int length() {
        return size();
    }

    /**
     * Returns the char value at the specified index. An index
     * ranges from zero to length() - 1. The first char
     * value of the sequence is at index zero, the next at index one, and so
on,
     * as for array indexing.
     *
     *
     * If the char value specified by the index is a
     * surrogate, the surrogate value is
     * returned.
     *
     * @param index
     *            the index of the char value to be returned
     * @return the specified char value
     * @throws IndexOutOfBoundsException
     *             if the index argument is negative or not less than
     *             length()
     */
    public char charAt(int index) {
        find0(index);
        return current.ch;
    }

    /**
     * Returns a new CharSequence that is a subsequence of this
     * sequence. The subsequence starts with the char value at
     * the specified index and ends with the char value at index
     * end - 1. The length (in chars) of the
     * returned sequence is end - start, so if start == end
     * then an empty sequence is returned.
     *
     *
     * @param start
     *            the start index, inclusive
     * @param end
     *            the end index, exclusive
     * @return the specified subsequence
     * @throws IndexOutOfBoundsException
     *             if start or end are negative, if
     *             end is greater than length(), or if
     *             start is greater than end
     */
    public CharSequence subSequence(int start, int end) {
        return new CharList(substring(start, end));
    }

    /**
     * The MAIN method, that allow high performance in this CharList, it
buffer
```

```
     * the current position to not recalculate it at any step
     * TODO this should calculate the best starting point when index !=
position
     *
     * @param index
     */
    private void find0(int index) {
        if (index == pos)
            // Use cache
            return;
        int start = index > pos ? pos : 0;
        current = index > pos ? current : head;
        for (int i = start; i < index; i++)
            current = current.getNext();
        pos = index;
    }
}

final class CharNode
        implements Serializable {

    private static final long serialVersionUID = -4603506751569436005L;

    protected CharNode next;

    protected final char ch;

    protected CharNode(char ch) {
        this.ch = ch;
    }

    protected void setNext(CharNode n) {
        this.next = n;
    }

    protected CharNode getNext() {
        return next;
    }

    protected char getChar() {
        return ch;
    }
}
```

4.3.2 *PropertyExt*

A Java Property files utility that supports sections and recursive properties

```
/**
 * Project: BIRO-Project (Funded by European Commission 2005-2008)
 *
 * File:     PropertyExt.java
 * Author:   Pietro Palladino
 *
 * License: TO Be Written. - Free and Open to All BIRO Partners
 *
 */
package org.plibrary.util;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Map.Entry;

/**
 *
 * Allow to read and save Extended Property file
```

```
 *
 * @author Pietro
 *
 */
public class PropertyExt {

    private static final String defaultSection = "DEFAULT";

    private final Map<String, Map<String, String>> map;
    private String currentSection = defaultSection;

    public PropertyExt() {
        map = new LinkedHashMap<String, Map<String, String>>();
    }

    public PropertyExt(InputStream in) throws IOException {
        this();
        readFrom(in);
    }

    public PropertyExt(File f) throws IOException {
        this();
        readFrom(f);
    }

    private Map<String, String> getPropertiesMap(String sectionName) {
        Map<String, String> m = map.get(sectionName);
        if (m == null)
            map.put(sectionName, m = new LinkedHashMap<String, String>());
        return m;
    }

    private Map<String, String> getPropertiesMap() {
        return getPropertiesMap(currentSection);
    }

    public Iterable<Entry<String, String>> getPropertiesInSection(String
sectionName) {
        return getPropertiesMap(sectionName).entrySet();
    }

    /**
     * Returns all properties in current section
     */
    public Iterable<Entry<String, String>> getProperties() {
        return getPropertiesMap().entrySet();
    }

    /**
     * Returns true if it has a property in this section
     */
    public boolean hasPropertyInSection(String name) {
        return getPropertiesMap().containsKey(name);
    }

    /**
     * Returns true if it has a property in any section
     */
    public boolean hasPropertyInAnySection(String name) {
        return findSectionOfProperty(name) != null;
    }

    /**
     * Returns property in current section
     */
    public String getProperty(String propName) {
        return getPropertiesMap().get(propName);
    }

    /**
     * Returns property in sepcified section
     */
    public String getPropertyInSection(String section, String propName) {
        return getPropertiesMap(section).get(propName);
    }
```

```
    /**
     * Returns derived property.
     * That is:
     * If property A has value B and property B has value C the call of
     * getDerivedProperty(A) returns C
     */
    public String getDerivedProperty(String name) {
        String section = findSectionOfProperty(name);
        String value = getPropertyInSection(section, name);
        return hasPropertyInAnySection(value) ? getDerivedProperty(value) :
value;
    }

    /**
     * Sets property in current section
     */
    public void setProperty(String name, String value) {
        getPropertiesMap().put(name, value);
    }

    /**
     * Change section
     */
    public void changeSection(String name) {
        currentSection = name;
    }

    /**
     * Goes to default section, the root
     *
     */
    public void defaultSection() {
        changeSection(defaultSection);
    }

    /**
     * Returns the name of the section containing the property
     */
    public String findSectionOfProperty(String name) {
        for (Entry<String, Map<String, String>> e1 : map.entrySet()) {
            for (Entry<String, String> e2 : e1.getValue().entrySet())
                if (e2.getKey().equals(name))
                    return e1.getKey();
        }
        return null;
    }

    public void readFrom(File f) throws IOException {
        readFrom(new FileInputStream(f));
    }

    public void readFrom(InputStream in) throws IOException {
        BufferedReader r = new BufferedReader(new InputStreamReader(in));
        try {
            String line = null;
            while ((line = r.readLine()) != null) {
                line = line.trim();
                if (line.length() == 0 || line.startsWith("#")) // Comment
                    continue;
                if (line.startsWith("[")) // Section
                    currentSection = line.substring(1, line.length() - 1);
                else {
                    String[] pv = line.split("=");
                    String name = pv[0];
                    String value = pv.length == 1 ? null : pv[1];
                    setProperty(name, value);
                }
            }
        }
        catch (IOException e) {
            throw e;
        }
        finally {
            r.close();
        }
    }
```

```
    public void writeTo(File f) throws IOException {
        writeTo(new FileOutputStream(f));
    }

    public void writeTo(OutputStream o) {
        CharList l = new CharList();

        for (Entry<String, Map<String, String>> entry : map.entrySet()) {
            if (!defaultSection.equals(entry.getKey())) {
                l.append('[').append(entry.getKey()).append(']').newLine();
                l.newLine();
            }
            for (Entry<String, String> e : entry.getValue().entrySet()) {
                l.append(e.getKey()).append('=').append(e.getValue());
                l.newLine();
            }
            l.newLine();
        }
        l.printString(o);
    }
}
```

4.3.3 *ProgressCalculator*

An utility to calculate and show a progress bar and estimated remaining time on standard output

```
/**
 * Project: BIRO-Project (Funded by European Commission 2005-2008)
 *
 * File:     ProgressCalculator.java
 * Author:   Pietro Palladino
 *
 * License: TO Be Written. - Free and Open to All BIRO Partners
 *
 */
package org.plibrary.util;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

public final class ProgressCalculator
        implements Runnable {

    private static final int ROTOR_WAIT = 300;
    private static final int bar_step = 3;
    private static final int numBar = 100 / bar_step;
    private static final char[] ROTORS = { '|', '/', '-', '\\' };

    private static final DateFormat hour_f = new
SimpleDateFormat("HH'h':mm'm':ss's'");
    private static final DateFormat min_f = new
SimpleDateFormat("mm'm':ss's'");
    private static final DateFormat sec_f = new SimpleDateFormat("ss's'");
    private static final DateFormat msec_f = new SimpleDateFormat("SS'ms'");

    private static int rotor_index = 0;

    private static String toPrint;
    private static Thread printer;
    private static boolean rotor_printing = false;

    private final Calendar start_time;
    private final Calendar elapsed;
    private final Calendar remaining;
    private final long MAX;
    private final String task_name;
    private final CharList line = new CharList();

    /** Steps and percs */
```

```
    private long step_current = 0;
    private int perc_delta = -1;
    private int perc_current = 0;
    private long speed = 0;
    private long last_speed_step = 0;
    private long last_speed_time = 0;

    private static final String getCorrectFormat(Calendar c) {
        Date d = c.getTime();
        if (c.get(Calendar.HOUR_OF_DAY) > 1)
            return hour_f.format(d);
        if (c.get(Calendar.MINUTE) > 0)
            return min_f.format(d);
        if (c.get(Calendar.SECOND) > 0)
            return sec_f.format(d);
        return msec_f.format(d);
    }

    private static final long now() {
        return System.currentTimeMillis();
    }

    public ProgressCalculator(String task_name, int max) {
        this.MAX = max;
        this.task_name = task_name;
        start_time = Calendar.getInstance();
        elapsed = Calendar.getInstance();
        elapsed.setTimeInMillis(start_time.getTimeInMillis());
        remaining = Calendar.getInstance();
        remaining.clear();
    }

    public ProgressCalculator(int max) {
        this("", max);
    }

    public ProgressCalculator(String task_name) {
        this(task_name, 0);
    }

    public ProgressCalculator() {
        this(0);
    }

    public void doStep() {
        setNextStep(step_current + 1);
    }

    public void setNextStep(long next_step) {
        if (MAX == 0)
            throw new IllegalStateException("Cannot step when MAX is set to
0!");
        step_current = next_step;
        calculatePerc();
        if (hasChanged())
            calculateSpeed();
        calculateRemaining();
    }

    public void done() {
        setNextStep(MAX);
    }

    private void calculatePerc() {
        int new_perc = Math.round((step_current * 100f) / MAX);
        perc_delta = new_perc - perc_current;
        perc_current = new_perc;
    }

    private void calculateSpeed() {
        double msec = now() - last_speed_time;
        if (msec == 0)
            msec = 1;
        long delta = step_current - last_speed_step;
        speed = Math.round((delta * 1000) / msec);
        last_speed_time = now();
```

```
        last_speed_step = step_current;
    }

    private void calculateElapsed() {
        elapsed.setTimeInMillis(now() - start_time.getTimeInMillis());
        elapsed.roll(Calendar.HOUR_OF_DAY, false);
    }

    private void calculateRemaining() {
        if (speed == 0)
            return;
        long to_go = (1000 * (MAX - step_current)) / speed;
        remaining.setTimeInMillis(to_go);
        remaining.roll(Calendar.HOUR_OF_DAY, false);
    }

    public boolean hasChanged() {
        return perc_delta != 0;
    }

    public long getSpeed() {
        return speed;
    }

    public String getElapsedTime() {
        calculateElapsed();
        return getCorrectFormat(elapsed);
    }

    public String getRemainingTime() {
        return getCorrectFormat(remaining);
    }

    public int getPercent() {
        return perc_current;
    }

    public long getLastStep() {
        return step_current;
    }

    public long getMaximum() {
        return MAX;
    }

    public boolean isCompleted() {
        return step_current == MAX;
    }

    public void printGraphicOutput() {
        clearLine();
        line.append(task_name);
        line.append(": |");
        int step = getPercent() / bar_step;
        int i;
        for (i = 0; i < step; i++)
            line.append("=");
        for (; i < numBar; i++)
            line.append(" ");
        line.append(isCompleted() ? "|" : getNextRotor()).append(" ");
        line.append(getPercent()).append("%");
        if (isCompleted())
            line.append(" - Time: ").append(getElapsedTime()).newLine();
        else
            line.append(" - ETA: ").append(getRemainingTime());
        line.append('\r');
        System.out.print(line.toString());
    }

    private void clearLine() {
        CharList l = new CharList();
        for (int i = 0; i < line.size(); i++)
            l.append(" ");
        System.out.print("\r" + l.toString() + "\r");
        line.clear();
    }
```

```
    public static char getNextRotor() {
        char r = ROTORS[rotor_index];
        rotor_index = (rotor_index + 1) % ROTORS.length;
        return r;
    }

    public void startPrintRotors(String toprint) {
        if (printer != null)
            throw new RuntimeException("Rotors are already printed!");
        toPrint = toprint;
        rotor_printing = true;
        printer = new Thread(this);
        printer.setDaemon(true);
        System.out.print("\r
\r");
        printer.start();
    }

    public void stopPrintRotors() {
        rotor_printing = false;
        if (printer != null)
            try {
                printer.join();
            }
            catch (InterruptedException ex) {}
        printer = null;
    }

    public void run() {
        while (rotor_printing) {
            System.out.print(toPrint + " " + getNextRotor() + "\r");
            try {
                Thread.sleep(ROTOR_WAIT);
            }
            catch (InterruptedException ex) {}
        }
        System.out.print("\r" + toPrint + "   ");
    }
}
```

5. BIRO Database Manager Source Code

5.1. BIROClassGenerator

```
/**
 * Project: BIRO-Project (Funded by European Commission 2005-2008)
 *
 * File:     BIROClassGenerator.java
 * Author:   Valentina Baglioni
 *
 * License: TO Be Written. - Free and Open to All BIRO Partners
 *
 */
package test;

import org.exolab.castor.builder.SourceGenerator;

/**
 * This simple class allow to read the BIRO XML schema and
 * to automatically generate Java classes.
 *
 * @author Valentina Baglioni
 */

public class BIROClassGenerator {
      /**
     * Execute BIROClassGenerator
     * It requires as arguments:
     * 0) Input File Name of the binding file
     * 1) Input File Name of the BIRO XML schema
     * 2) Output Directory where Java classes will be stored
     */
       public static void main(String[] args) {
             try {
                    if (args.length < 3) {
                   System.err.println("\nUsage:\n\t" +
BIROClassGenerator.class.getName()
                                + " [bindingFile] [XMLschema] [outputDirectory]");
                   return;
              }

                    SourceGenerator sourceGen = new SourceGenerator();

                    String bindingFile = args[0];
                     String XMLschema = args[1];
                     String outputDirectory = args[2];

                     sourceGen.setBinding(bindingFile);

                     sourceGen.generateSource(XMLschema,outputDirectory);

             } catch (Exception e) {
                    System.out.println(e.toString());
             }

      }
}
```

5.2. Binding file

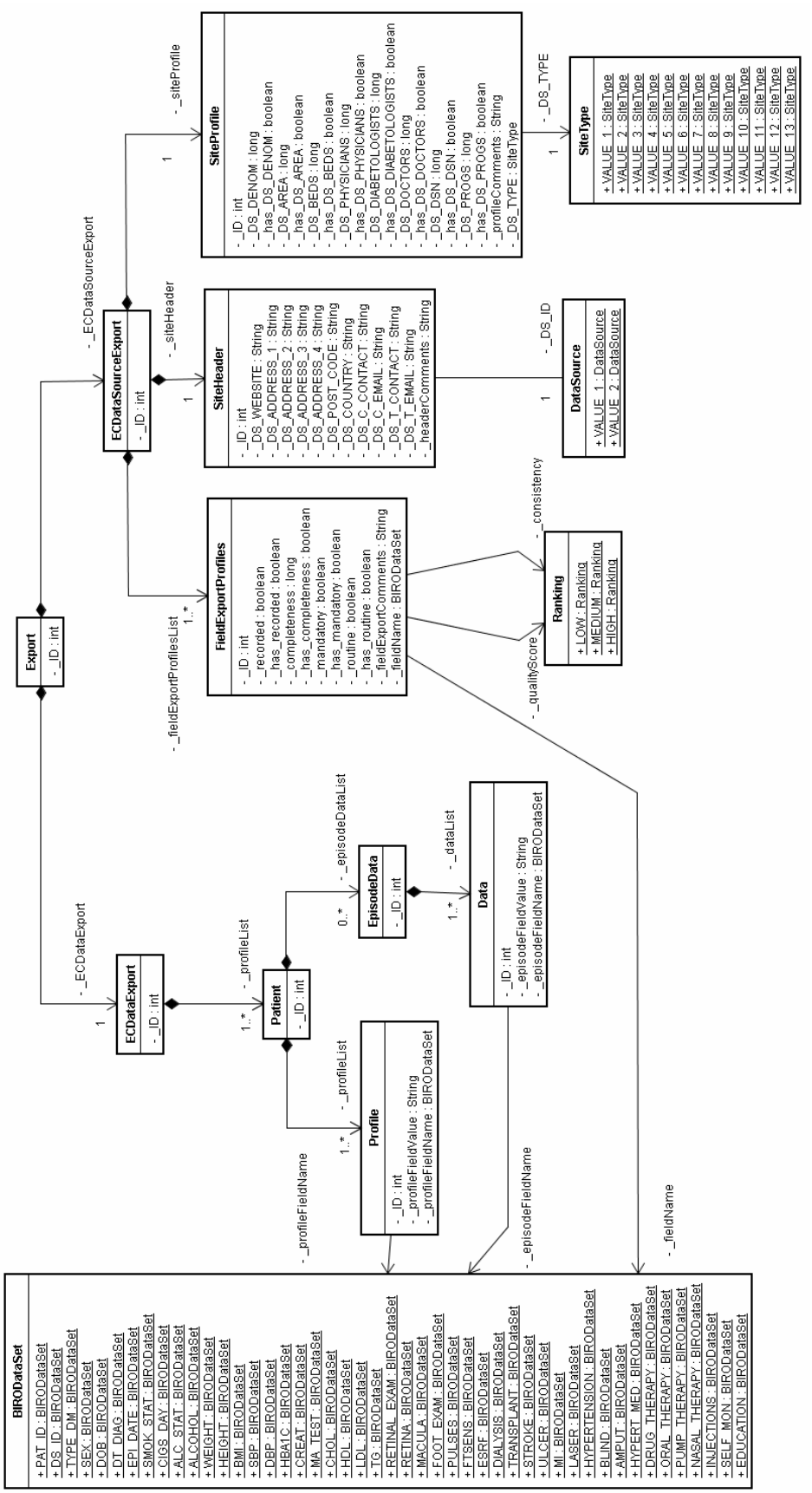
```
<binding xmlns="http://www.castor.org/SourceGenerator/Binding"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xxsi:noNamespaceSchemaLocation="C://Documents and
Settings/Valentina/.../binding.xsd"
         defaultBinding="element">

   <elementBinding name="/Export/ECDataSourceExport/FieldExportProfiles">
```

```
      <member collection="arraylist"/>
   </elementBinding>
   <elementBinding name="/Export/ECDataExport/Patient">
      <member collection="arraylist"/>
   </elementBinding>
   <elementBinding name="/Export/ECDataExport/Patient/Profile">
      <member collection="arraylist"/>
   </elementBinding>
   <elementBinding name="/Export/ECDataExport/Patient/EpisodeData">
      <member collection="arraylist"/>
   </elementBinding>
   <elementBinding name="/Export/ECDataExport/Patient/EpisodeData/Data">
      <member collection="arraylist"/>
   </elementBinding>

</binding>
```

5.3. Java Object Model UML diagram



5.4. Mapping files

5.4.1 BIRODataSet.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="export.types.BIRODataSet" table="BIRODATASET">
        <id name="type" type="string" column="FIELD_ID">
        </id>
        <property name="stringValue" type="string" column="FIELD_NAME"/>
    </class>
</hibernate-mapping>
```

5.4.2 Data.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="export.Data" table="DATA">
        <id name="ID" column="DATA_ID">
            <generator class="native"/>
        </id>
        <property name="EpisodeFieldName"
type="export.types.BIRODataSetUserType" column="EPISODE_FIELD_NAME"/>
        <property name="EpisodeFieldValue" type="string"
column="EPISODE_FIELD_VALUE"/>
</class>
</hibernate-mapping>
```

5.4.3 ECDataExport.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="export.ECDataExport" table="EC_DATA_EXPORT" >
        <id name="ID" column="EC_DATA_EXPORT_ID">
            <generator class="native"/>
        </id>
        <bag name="PatientList" order-by="PATIENT_ID" cascade="all">
            <key column="EC_DATA_EXPORT_ID" />
            <one-to-many class="export.Patient" />
        </bag>
    </class>
</hibernate-mapping>
```

5.4.4 ECDataSourceExport.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="export.ECDataSourceExport" table="EC_DATA_SOURCE_EXPORT" >
        <id name="ID" column="EC_DATA_SOURCE_EXPORT_ID">
            <generator class="native"/>
        </id>
        <many-to-one name="siteHeader"
                        column="SITE_HEADER_ID"
```

```
                            unique="true"
                            cascade="all"/>
        <many-to-one name="SiteProfile"
                            column="SITE_PROFILE_ID"
                            unique="true"
                            cascade="all"/>
        <bag name="fieldExportProfilesList" order-
by="FIEL_EXPORT_PROFILES_ID" cascade="all">
              <key column="EC_DATA_SOURCE_EXPORT_ID" />
              <one-to-many class="export.FieldExportProfiles" />
        </bag>
    </class>
</hibernate-mapping>
```

5.4.5 EpisodeData.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="export.EpisodeData" table="EPISODE_DATA" >
        <id name="ID" column="EPISODE_DATA_ID">
              <generator class="native"/>
        </id>
        <property name="hibernateEpisodeDate" type="date"
column="EPISODE_DATE"/>
        <bag name="DataList" order-by="DATA_ID" cascade="all">
              <key column="EPISODE_DATA_ID" />
              <one-to-many class="export.Data" />
        </bag>
    </class>
</hibernate-mapping>
```

5.4.6 Export.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="export.Export" table="EXPORT" >
        <id name="ID" column="EXPORT_ID">
              <generator class="native"/>
        </id>
        <many-to-one name="ECDataExport"
                            column="EC_DATA_EXPORT_ID"
                            unique="true"
                            cascade="all"/>
        <many-to-one name="ECDataSourceExport"
                            column="EC_DATA_sOURCE_EXPORT_ID"
                            unique="true"
                            cascade="all"/>
    </class>
</hibernate-mapping>
```

5.4.7 FieldExportProfiles.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="export.FieldExportProfiles" table="FIELD_EXPORT_PROFILES">
        <id name="ID" column="FIELD_EXPORT_PROFILES_ID">
              <generator class="native"/>
        </id>
        <property name="fieldName" type="export.types.BIRODataSetUserType"
column="FIELD_NAME"/>
        <property name="hibernateDateStatusLastReviewed" type="date"
column="DATE_STATUS_LAST_REVIEWED"/>
```

```
        <property name="recorded" type="boolean" column="RECORDED"/>
        <property name="consistency" type="export.types.RankingUserType"
column="CONSISTENCY" not-null="false"/>
        <property name="completeness" type="long" column="COMPLETENESS"/>
        <property name="mandatory" type="boolean" column="MANDATORY"/>
        <property name="routine" type="boolean" column="ROUTINE"/>
        <property name="qualityScore" type="export.types.RankingUserType"
column="QUALITY_SCORE"/>
        <property name="fieldExportComments" type="string"
column="FIELD_EXPORT_COMMENTS"/>
</class>
</hibernate-mapping>
```

5.4.8 Patient.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="export.Patient" table="PATIENT" >
        <id name="ID" column="PATIENT_ID">
            <generator class="native"/>
        </id>
        <bag name="ProfileList" order-by="PROFILE_ID" cascade="all">
            <key column="PATIENT_ID" />
            <one-to-many class="export.Profile" />
        </bag>
        <bag name="EpisodeDataList" order-by="EPISODE_DATA_ID"
cascade="all">
            <key column="PATIENT_ID" />
            <one-to-many class="export.EpisodeData" />
        </bag>
    </class>
</hibernate-mapping>
```

5.4.9 Profile.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="export.Profile" table="PROFILE">
        <id name="ID" column="PROFILE_ID">
            <generator class="native"/>
        </id>
        <property name="ProfileFieldName"
type="export.types.BIRODataSetUserType" column="PROFILE_FIELD_NAME"/>
        <property name="ProfileFieldValue" type="string"
column="PROFILE_FIELD_VALUE"/>
</class>
</hibernate-mapping>
```

5.4.10 SiteHeader.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="export.SiteHeader" table="SITE_HEADER">
        <id name="ID" column="SITE_HEADER_ID">
            <generator class="native"/>
        </id>
        <property name="hibernateDateHeaderInformationChecked" type="date"
column="DATE_HEADER_INFORMATION_CHECKED"/>
        <property name="DS_ID" type="export.types.DataSourceUserType"
column="DS_ID"/>
        <property name="DS_WEBSITE" type="string" column="DS_WEBSITE"/>
```

```
        <property name="DS_ADDRESS_1" type="string" column="DS_ADDRESS_1"/>
        <property name="DS_ADDRESS_2" type="string" column="DS_ADDRESS_2"/>
        <property name="DS_ADDRESS_3" type="string" column="DS_ADDRESS_3"/>
        <property name="DS_ADDRESS_4" type="string" column="DS_ADDRESS_4"/>
        <property name="DS_POST_CODE" type="string" column="DS_POST_CODE"/>
        <property name="DS_COUNTRY" type="string" column="DS_COUNTRY"/>
        <property name="DS_C_CONTACT" type="string" column="DS_C_CONTACT"/>
        <property name="DS_C_EMAIL" type="string" column="DS_C_EMAIL"/>
        <property name="DS_T_CONTACT" type="string" column="DS_T_CONTACT"/>
        <property name="DS_T_EMAIL" type="string" column="DS_T_EMAIL"/>
        <property name="headerComments" type="string"
column="HEADER_COMMENTS"/>
</class>
</hibernate-mapping>
```

5.4.11 SiteProfile.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="export.SiteProfile" table="SITE_PROFILE">
        <id name="ID" column="SITE_PROFILE_ID">
            <generator class="native"/>
        </id>
        <property name="hibernateDateProfileInformationChecked" type="date"
column="DATE_PROFILE_INFORMATION_CHECKED"/>
        <property name="DS_TYPE" type="export.types.SiteTypeUserType"
column="DS_TYPE"/>
        <property name="DS_DENOM" type="long" column="DS_DENOM"/>
        <property name="DS_AREA" type="long" column="DS_AREA"/>
        <property name="DS_BEDS" type="long" column="DS_BEDS"/>
        <property name="DS_PHYSICIANS" type="long" column="DS_PHYSICIANS"/>
        <property name="DS_DIABETOLOGISTS" type="long"
column="DS_DIABETOLOGISTS"/>
        <property name="DS_DOCTORS" type="long" column="DS_DOCTORS"/>
        <property name="DS_DSN" type="long" column="DS_DSN"/>
        <property name="DS_PROGS" type="long" column="DS_PROGS"/>
        <property name="ProfileComments" type="string"
column="PROFILE_COMMENTS"/>
</class>
</hibernate-mapping>
```

5.5. BIRO Database structure

5.5.1 SQL code: table “data”

```
CREATE TABLE data
(
  data_id integer NOT NULL,
  episode_field_name character varying(255),
  episode_field_value character varying(255),
  episode_data_id integer,
  CONSTRAINT data_pkey PRIMARY KEY (data_id),
  CONSTRAINT fk1fe7aa13e9d477 FOREIGN KEY (episode_data_id)
      REFERENCES episode_data (episode_data_id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITHOUT OIDS;
ALTER TABLE data OWNER TO "BIROdatabase";
```

5.5.2 SQL code: table “ec_data_export”

```
CREATE TABLE ec_data_export
(
  ec_data_export_id integer NOT NULL,
  CONSTRAINT ec_data_export_pkey PRIMARY KEY (ec_data_export_id)
)
WITHOUT OIDS;
```

```
ALTER TABLE ec_data_export OWNER TO "BIROdatabase";
```

5.5.3 SQL code: table "ec_data_source_export"

```
CREATE TABLE ec_data_source_export
(
  ec_data_source_export_id integer NOT NULL,
  site_header_id integer,
  site_profile_id integer,
  CONSTRAINT ec_data_source_export_pkey PRIMARY KEY
(ec_data_source_export_id),
  CONSTRAINT fkfc1a79c41e396871 FOREIGN KEY (site_profile_id)
      REFERENCES site_profile (site_profile_id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fkfc1a79c4d7794063 FOREIGN KEY (site_header_id)
      REFERENCES site_header (site_header_id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT ec_data_source_export_site_header_id_key UNIQUE
(site_header_id),
  CONSTRAINT ec_data_source_export_site_profile_id_key UNIQUE
(site_profile_id)
)
WITHOUT OIDS;
ALTER TABLE ec_data_source_export OWNER TO "BIROdatabase";
```

5.5.4 SQL code: table "episode_data"

```
CREATE TABLE episode_data
(
  episode_data_id integer NOT NULL,
  episode_date date,
  patient_id integer,
  CONSTRAINT episode_data_pkey PRIMARY KEY (episode_data_id),
  CONSTRAINT fkec38478e322a9f40 FOREIGN KEY (patient_id)
      REFERENCES patient (patient_id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITHOUT OIDS;
ALTER TABLE episode_data OWNER TO "BIROdatabase";
```

5.5.5 SQL code: table "export"

```
CREATE TABLE export
(
  export_id integer NOT NULL,
  ec_data_export_id integer,
  ec_data_source_export_id integer,
  CONSTRAINT export_pkey PRIMARY KEY (export_id),
  CONSTRAINT fk7abc07b428585327 FOREIGN KEY (ec_data_source_export_id)
      REFERENCES ec_data_source_export (ec_data_source_export_id) MATCH
SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk7abc07b4f7a0fc28 FOREIGN KEY (ec_data_export_id)
      REFERENCES ec_data_export (ec_data_export_id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT export_ec_data_export_id_key UNIQUE (ec_data_export_id),
  CONSTRAINT export_ec_data_source_export_id_key UNIQUE
(ec_data_source_export_id)
)
WITHOUT OIDS;
ALTER TABLE export OWNER TO "BIROdatabase";
```

5.5.6 SQL code: table "field_export_profiles"

```
CREATE TABLE field_export_profiles
(
  field_export_profiles_id integer NOT NULL,
  field_name character varying(255),
```

```
  date_status_last_reviewed date,
  recorded boolean,
  consistency character varying(255),
  completeness bigint,
  mandatory boolean,
  routine boolean,
  quality_score character varying(255),
  field_export_comments character varying(255),
  ec_data_source_export_id integer,
  CONSTRAINT field_export_profiles_pkey PRIMARY KEY
(field_export_profiles_id),
  CONSTRAINT fk2423ab028585327 FOREIGN KEY (ec_data_source_export_id)
      REFERENCES ec_data_source_export (ec_data_source_export_id) MATCH
SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITHOUT OIDS;
ALTER TABLE field_export_profiles OWNER TO "BIROdatabase";
```

5.5.7 SQL code: table “patient”

```
CREATE TABLE patient
(
  patient_id integer NOT NULL,
  ec_data_export_id integer,
  CONSTRAINT patient_pkey PRIMARY KEY (patient_id),
  CONSTRAINT fkfb9f76e5f7a0fc28 FOREIGN KEY (ec_data_export_id)
      REFERENCES ec_data_export (ec_data_export_id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITHOUT OIDS;
ALTER TABLE patient OWNER TO "BIROdatabase";
```

5.5.8 SQL code: table “profile”

```
CREATE TABLE profile
(
  profile_id integer NOT NULL,
  profile_field_name character varying(255),
  profile_field_value character varying(255),
  patient_id integer,
  CONSTRAINT profile_pkey PRIMARY KEY (profile_id),
  CONSTRAINT fk185a1589322a9f40 FOREIGN KEY (patient_id)
      REFERENCES patient (patient_id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITHOUT OIDS;
ALTER TABLE profile OWNER TO "BIROdatabase";
```

5.5.9 SQL code: table “site_header”

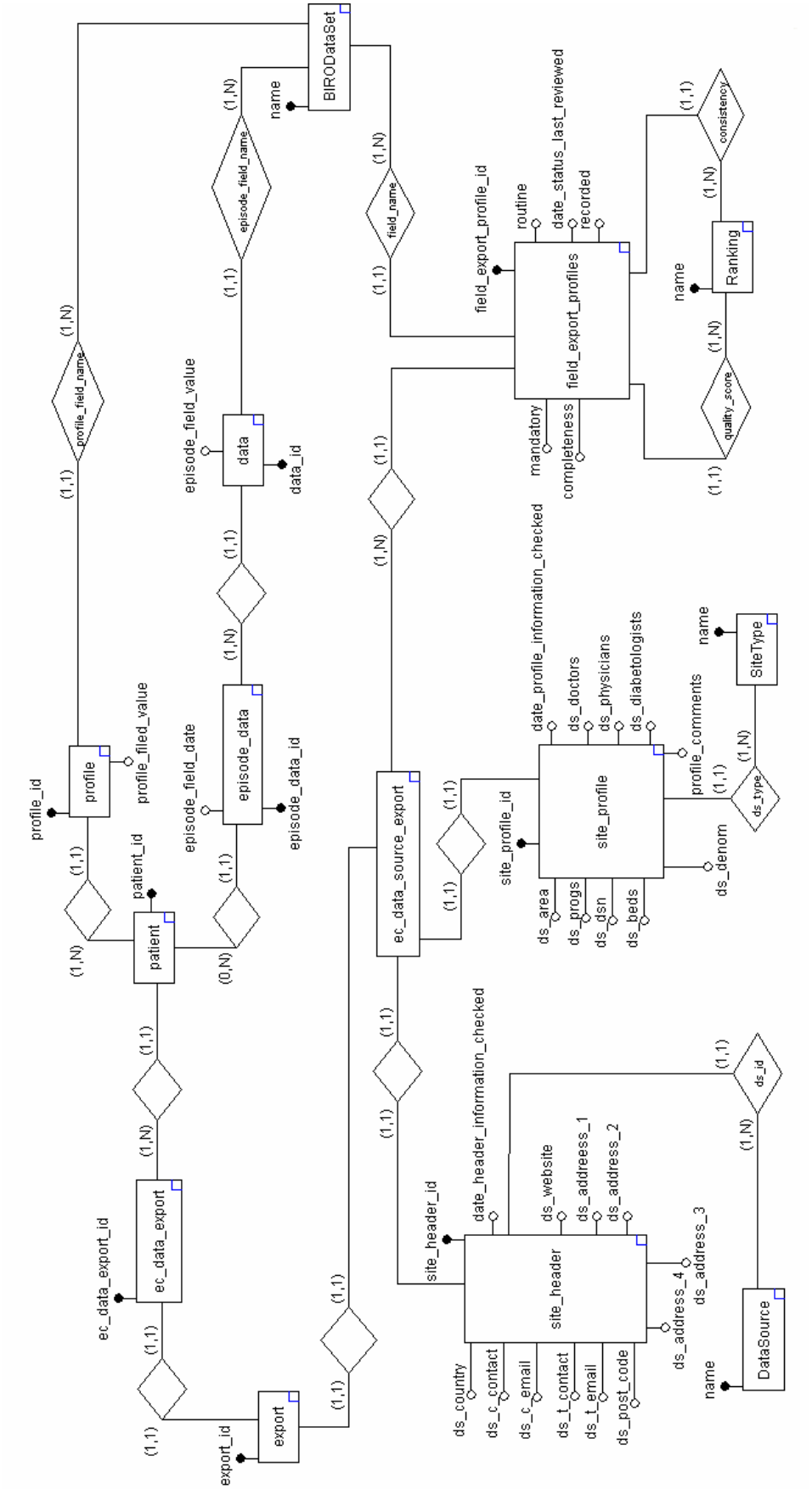
```
CREATE TABLE site_header
(
  site_header_id integer NOT NULL,
  date_header_information_checked date,
  ds_id character varying(255),
  ds_website character varying(255),
  ds_address_1 character varying(255),
  ds_address_2 character varying(255),
  ds_address_3 character varying(255),
  ds_address_4 character varying(255),
  ds_post_code character varying(255),
  ds_country character varying(255),
  ds_c_contact character varying(255),
  ds_c_email character varying(255),
  ds_t_contact character varying(255),
  ds_t_email character varying(255),
  header_comments character varying(255),
  CONSTRAINT site_header_pkey PRIMARY KEY (site_header_id)
)
```

```
WITHOUT OIDS;
ALTER TABLE site_header OWNER TO " BIROdatabase";
```

5.5.10 SQL code: table “site_profile”

```
CREATE TABLE site_profile
(
  site_profile_id integer NOT NULL,
  date_profile_information_checked date,
  ds_type character varying(255),
  ds_denom bigint,
  ds_area bigint,
  ds_beds bigint,
  ds_physicians bigint,
  ds_diabetologists bigint,
  ds_doctors bigint,
  ds_dsn bigint,
  ds_progs bigint,
  profile_comments character varying(255),
  CONSTRAINT site_profile_pkey PRIMARY KEY (site_profile_id)
)
WITHOUT OIDS;
ALTER TABLE site_profile OWNER TO "BIROdatabase";
```

5.5.11 ER diagram



5.6. BIRODatabaseManagerMain

```
/**
 * Project: BIRO-Project (Funded by European Commission 2005-2008)
 *
 * File:     DatabaseManagerMain.java
 * Author:   Valentina Baglioni
 *
 * License: TO Be Written. - Free and Open to All BIRO Partners
 *
 */

package test;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FilenameFilter;

import util.HibernateUtil;
import util.Loader;
import export.ECDataExport;
import export.ECDataSourceExport;

/**
 * This simple class allow one partner to read XML files following the BIRO
XML schema,
 * and store data into the local BIRO Database.(see documentation)
 *
 * @author Valentina Baglioni
 */
public final class BIRODatabaseManagerMain {

      /**
     * Execute BIRO Database Manager
     * It requires as arguments:
     * 0) Input File Name of the XML Configuration
     * 1) Input Directory of the BIRO Xml files
     */
      public static void main(String[] args) throws Exception {
            if (args.length < 2) {
           System.err.println("\nUsage:\n\t" +
BIRODatabaseManagerMain.class.getName() + " [ConfigFile] [OutputDir]");
            return;
       }

             Loader.addDir("lib");

             String configurationFileName = args[0];
             String directory = args[1];
             File configurationFile = new File(configurationFileName);
       if (!configurationFile.exists())
             throw new FileNotFoundException("Cannot find the Configuration
file at '"  + configurationFileName + "'");
             HibernateUtil.setSessionFactory(configurationFile);


             class ECDataExportFilter implements FilenameFilter {

                   public boolean accept(File dir, String name) {
                         return( (name.substring(0, name.length()-
4)).matches("[0-9]*") && name.endsWith(".xml"));
                   }
             }
             class ECDataSourceExportFilter implements FilenameFilter {

                   public boolean accept(File dir, String name) {
                         return (name.startsWith("ECDataSourceExport") &&
name.endsWith(".xml"));
                   }
             }

             for (String s : new File(directory).list(new
ECDataExportFilter()))
             {
                   System.out.println("\tMarshaling file " + s);
```

```
		UnmarshallingAndStoringManager.unmarshallAndStore(ECDataExport.class,
directory + "/" + s);
                        System.gc();
                }

                for (String s : new File(directory).list(new
ECDataSourceExportFilter()))
                {
                        System.out.println("\tMarshaling file " + s);

		UnmarshallingAndStoringManager.unmarshallAndStore(ECDataSourceExport.c
lass,  directory + "/" + s);
                        System.gc();
                }
        }

}
```

5.7. UnmarshallingAndStoringManager

```
/**
 * Project: BIRO-Project (Funded by European Commission 2005-2008)
 *
 * File:     UnmarshallingAndStoringManager.java
 * Author:   Valentina Baglioni
 *
 * License: TO Be Written. - Free and Open to All BIRO Partners
 *
 */

package test;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;

import org.exolab.castor.mapping.MappingException;
import org.exolab.castor.xml.MarshalException;
import org.exolab.castor.xml.Unmarshaller;
import org.exolab.castor.xml.ValidationException;
import org.hibernate.Session;

import util.HibernateUtil;

/**
 * This simple class allow to unmarshall and store a BIRO Export XML file
 *
 * @author Valentina Baglioni
 */


public class UnmarshallingAndStoringManager {

    /**
     * Unmarshall and store a BIRO Export XML File
     * @param c
     *              class to be unmarshalled
     * @param file
     *              string representing the file to be unmarshalled
     */

        public static void unmarshallAndStore(Class<?> c, String file) throws
IOException,
                        MappingException, MarshalException, ValidationException
{
                FileReader reader = new FileReader(file);
                Object o = unmarshall(c, reader);
                store(o);
        }

        private static Object unmarshall(Class<?> c, FileReader reader) throws
MappingException,
                        MarshalException, ValidationException {
```

```
            return Unmarshaller.unmarshal(c, reader);
      }

      private static void store(Object o) {
            Session session2 =
HibernateUtil.getSessionFactory().getCurrentSession();
            session2.beginTransaction();
            session2.persist(o);
            session2.getTransaction().commit();
            HibernateUtil.getSessionFactory().close();
      }
}
```

5.8. Utilities

5.8.1 HibernateUtil

```
/**
 * Project: BIRO-Project (Funded by European Commission 2005-2008)
 *
 * File:     HibernateUtil.java
 * Author:   Valentina Baglioni
 *
 * License: TO Be Written. - Free and Open to All BIRO Partners
 *
 */
package util;

import java.io.File;
import java.io.FileInputStream;
import java.util.Properties;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

/**
 * This simple class allow Hibernate to read the Configuration File where
 * database connection settings are stored
 *
 * @author Valentina Baglioni
 */

public class HibernateUtil {

      private static SessionFactory sessionFactory;
      private static File configurationFile;

    /**
     * Set a session factory
     * @param configFile
     *            File representing the configuration file
     *
     */

      public static void setSessionFactory(File configFile) {
            configurationFile = configFile;
            try {

                  Properties properties = new Properties();
                  properties.load(new FileInputStream(configurationFile));

                  Configuration cfg = new Configuration();

                  // Database connection settings are stored in the
configuration file
                  cfg.setProperties(properties);

                  // JDBC connection pool (use the built-in)
                  cfg.setProperty("hibernate.connection.pool_size", "1");

                  // Enable Hibernate's automatic session context
management
                 cfg.setProperty("hibernate.current_session_context_class",
"thread");
```

```
                    // Disable the second-level cache
                    cfg.setProperty("hibernate.cache.provider_class",
"org.hibernate.cache.NoCacheProvider");


                    // Echo all executed SQL to stdout
                    cfg.setProperty("hibernate.show_sql", "false");
                    // Drop and re-create the database schema on startup
                    cfg.setProperty("hibernate.hbm2ddl.auto", "create");


      cfg.addResource("hibernateMappings/ECDataSourceExport.hbm.xml");
      cfg.addResource("hibernateMappings/SiteHeader.hbm.xml");
      cfg.addResource("hibernateMappings/SiteProfile.hbm.xml");
      cfg.addResource("hibernateMappings/FieldExportProfiles.hbm.xml");
      cfg.addResource("hibernateMappings/BIRODataSet.hbm.xml");
      cfg.addResource("hibernateMappings/ECDataExport.hbm.xml");
      cfg.addResource("hibernateMappings/Patient.hbm.xml");
      cfg.addResource("hibernateMappings/Profile.hbm.xml");
      cfg.addResource("hibernateMappings/EpisodeData.hbm.xml");
      cfg.addResource("hibernateMappings/Data.hbm.xml");
      cfg.addResource("hibernateMappings/Export.hbm.xml");

                        sessionFactory = cfg.buildSessionFactory();

            } catch (Throwable ex) {
                    System.err.println("Initial SessionFactory creation
failed." + ex);
                    throw new ExceptionInInitializerError(ex);
            }

      }
      /**
     * return the session factory
     */
      public static SessionFactory getSessionFactory() {
            return sessionFactory;
      }
}
```

5.8.2 Loader

```
/**
 * Project: BIRO-Project (Funded by European Commission 2005-2008)
 *
 * File:     Loader.java
 * Author:   Pietro Palladino
 *
 * License: TO Be Written. - Free and Open to All BIRO Partners
 *
 */

package util;

import java.io.File;
import java.io.FilenameFilter;
import java.io.IOException;
import java.lang.reflect.Method;
import java.net.URL;
import java.net.URLClassLoader;

/**
 * Utility class for jar resources.
 */
public final class Loader {

      private Loader() {
      }

      /**
       * Add a dir to classpath loading Jar and Natives
       *
       * @param dir
       */
```

```
	public static void addDir(String dir) {
		if (dir == null)
			return;

		final class JarFilter implements FilenameFilter {

			public boolean accept(File dir, String name) {
				return name.endsWith(".jar");
			}
		}

		File d = new File(dir);
		if (!d.exists())
			return;

		FilenameFilter jar = new JarFilter();

		for (File f : d.listFiles(jar))
			try {
				loadJarFile(f);
			} catch (IOException e) {
			}
	}

	/**
	 * Add jar to classpath when out of classpath.
	 */
	private static void addURL(URL u) throws IOException {
		URLClassLoader sysloader = (URLClassLoader)
ClassLoader.getSystemClassLoader();
		Class<?> sysclass = URLClassLoader.class;
		try {
			Method method = sysclass.getDeclaredMethod("addURL",
URL.class);
			method.setAccessible(true);
			method.invoke(sysloader, u);
		} catch (Throwable t) {
			throw new IOException("Error, could not add URL to
system classloader:\n"
					+ t.getMessage());
		}
	}

	/**
	 * Add jar to classpath when out of classpath.
	 */
	public static void loadJarFile(File f) throws IOException {
		addURL(f.toURI().toURL());
	}

	/**
	 * Load native library.
	 */
	public static void loadNativeLib(String path, String name) {
		String full_name = (path == null ? "." : path) + "/" + name +
".dll";
		System.load(new File(full_name).getAbsolutePath());
	}
}
```

6. References

[1] Castor web site, http://www.castor.org/

[2] Castor - binding file, http://www.castor.org/srcgen-binding.html

[3] Hibernate from Wikipedia, http://en.wikipedia.org/wiki/Hibernate_(Java)

[4] Hibernate reference documentation version 3.2.2, http://www.hibernate.org/hib_docs/reference/en/html/index.html

[5] Hibernate web site, http://www.hibernate.org/

[6] PostgreSQL web site, http://www.postgresql.org/

[7] M. Birbeck, J. Duckett, O. G. Gudmundsson, P. Kobak, E. Lenz, S. Livingstone, D. Marcus, S. Mohr, N. Ozu, J. Pinnock, K. Visco, A. Watt, K. Williams, Z. Zaev, “Professional XML”, Wrox, May 2001.