The developer guide to the Data Wrapper

Hongtao Ren, Marek Makowski, IIASA

Version 0.1, February 15, 2012

1 Scope of this document

This guide aims at supporting the developers in using the data wrapper prototype (further on referred to as the wrapper), and in adapting the prototype to more specific needs. The document deals with the following issues:

- Functionality
- Architecture
- Structure and content of the part of the github repository containing the wrapper
- Compilation of the wrapper
- Installation of the wrapper at a site
- Integration of the wrapper with the test GUI

2 Functionality of the data wrapper prototype

The wrapper is a simple application developed for serving the following purposes:

- Illustrate how easy is to consume the WSs (Web-services)
- Provide a template for other applications consuming WSs for data management
- Provide actual functionality of uploading model components: members of sets, parameters, and variables.

Short characteristics of the wrapper:

- written in Java
- takes arguments from the cmd-line
- reads data from a CSV file
- uses a WS to get the SMS
- uses WSs for uploading data to the DW (DataWarehouse)

The wrapper prototype reads data from a CSV file; it can however be easily adapted for getting data from other sources, e.g., meters, local data bases, etc. Moreover, it also can easily use the already implemented functionality for reading data from XML files (see an example in the Kernel repository[1]).

[1] https://github.com/enrima-dev4/kernel/blob/master/clients/data_wrapper/wrapper/src/main/resources/demand.xml

The wrapper currently uses a self-documented CSV file (an example is available in the Kernel repository[2]). The currently implemented structure of the CSV files is as follows:

- lines starting with the # character as well as empty lines are treated as comments and therefore ignored during parsing
- first non-comment line serves as a header containing the entities' short names corresponding to the data specified in the lines that follow the header.

The wrapper is executed either from a command prompt or by a system call, and takes the following four arguments:

1. action type: takes one of the two values (entity, set) indicating whether the values read will correspond to the specified entities (parameters or variables) or members of sets
2. the CSV file name
3. id of the model
4. id of the data version

For example, the command: *wrapper entity demand.csv 71 23* will execute the wrapper for reading the values of parameters and/or variables from the file called demand.csv, for the model and data ids equal to 71 and 23, respectively.

3 Overview the architecture

Data wrapper can be installed at a site or can be seamlessly integrated into a GUI. Figure 1 shows the WS-based interface between the Kernel and the Wrapper.

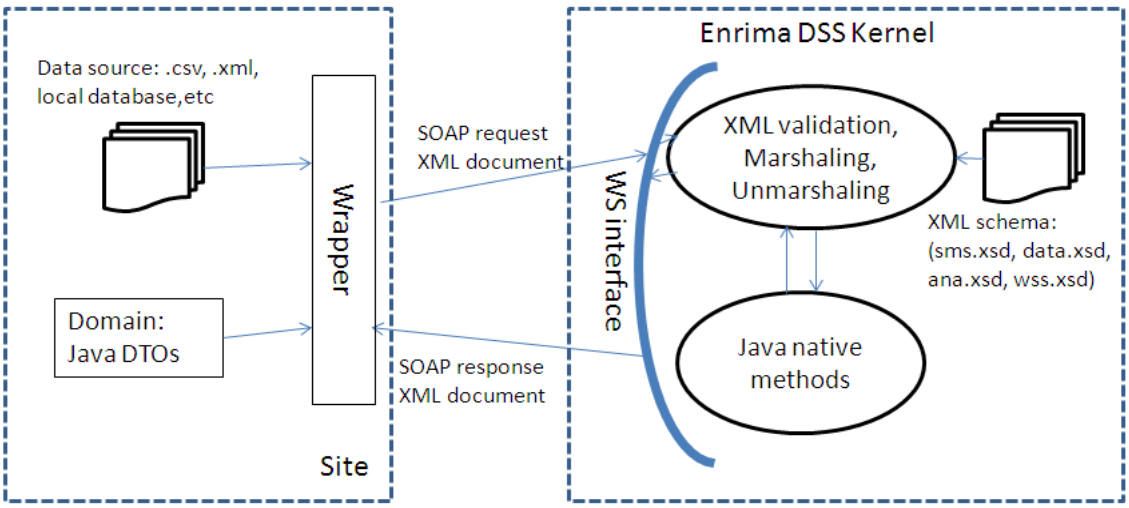

Figure 1 Kernel and wrapper interface

The Wrapper contains the domain module and the wrapper modules; the functionality of each module:

[2] https://github.com/enrima-dev4/kernel/blob/master/clients/data_wrapper/wrapper/src/main/resources/demand.csv

1) Domain: contains java classes (the structure of data transfer objects (DTOs)) which based on the contract (WSDL file), these classes are generated automatically through an Object/XML mapping (OXM) tool (Maven JAX-WS plugin).
2) Wrapper: organizes the read data into DTOs and then stores the DTOs' content into the Kernel.

4 Files and directories in each module

4.1 Domain module

- domain/pom.xml
  The Apache Maven (http://maven.apache.org/) project configuration file contains jaxws-maven-plugin (http://mojo.codehaus.org/jaxws-maven-plugin/wsimport-mojo.html) which can generate Java DTOs automatically based on a specification available in the WSDL file. The Figure2 contains a snippet from the pom.xml (posted at the Kernel repository) file illustrating the jaxws plugin using the WSDL from the URL: http://www.ime.iiasa.ac.at/enrima_ws_tst1/enrima.wsdl and generating the DTOs into the package "at.ac.iiasa.ime.enrima.client.domain"

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>jaxws-maven-plugin</artifactId>
    <executions>
     <execution>
       <goals>
         <goal>wsimport</goal>
       </goals>
       <configuration>
         <wsdlUrls>
           <wsdlUrl>http://www.ime.iiasa.ac.at/enrima_ws_tst1/enrima.wsdl</wsdlUrl>
         </wsdlUrls>
     <packageName>at.ac.iiasa.ime.enrima.client.domain</packageName>
       </configuration>
     </execution>
    </executions>
</plugin>
```

Figure2 code snippet of the pom.xml

- domain/target/jaxws/wsimport/java
  The automatically generated Java DTOs corresponding to the XML schemas described in the http://www.ime.iiasa.ac.at/enrima_ws_tst1/enrima.wsdl, the Figure3 is the code snippet of domain\target\jaxws\wsimport\java\at\ac\iiasa\ime\enrima\client\domain\ModelSpec.java is generated automatically based on the XML Data type of the modelSpec described in the WSDL file.

```
package at.ac.iiasa.ime.enrima.client.domain;
 ...
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "modelSpec", namespace = "http://www.ime.iiasa.ac.at/model/spec", propOrder = {
    "id",
    "shortName",
    "name",
    "description",
    "status",
    "setSpec",
    "entitySpec"
})
public class ModelSpec {
    protected Integer id;
    @XmlElement(required = true)
    protected String shortName;
    protected String name;
    protected String description;
    @XmlElement(required = true)
    protected Status status;
    @XmlElement(required = true)
    protected List<SetSpec> setSpec;
    @XmlElement(required = true)
    protected List<EntitySpec> entitySpec;
    ...
}
```

Figure3 code snippet of the ModelSpec.java

4.2 Wrapper module

- wrapper/pom.xml
  It contains the project dependencies can be found within the <dependencies> </dependencies>. We can see the wrapper mainly depends on the domain and the Spring Web Services (http://static.springsource.org/spring-ws/sites/2.0/). It also describes the settings for generating executable jar file by using maven-jar-plugin (http://maven.apache.org/plugins/maven-jar-plugin/) and onejar-maven-plugin (http://onejar-maven-plugin.googlecode.com/svn/mavensite/usage.html).
- directory: wrapper/src/main/java/at/ac/iiasa/ime/enrima/client/datawrapper
  - EnrimaClient.java is an abstract class extends Spring WebServiceGatewaySupport which can send/return DTOs to/from the Enrima DSS Kernel.
  - DataWrapperClient.java is the interface which declares all the functions.

```
import at.ac.iiasa.ime.enrima.client.domain.ModelSpec;

public interface DataWrapperClient {
    public ModelSpec getModelById(int idModel);
    public String consumeWS(File xmlFile);
    public String uploadEntitiesValues(String filePath, int idModel,int idModelData)throws IOException;
    public String uploadMainSet(String filePath, int idModel,int idModelData)throws IOException;
}
```

Figure 4 the code snippet of the DataWrapperClient.java

  - DataWrapperClientImpl.java is the implement of the interface DataWrapperClient, the Figure 5 shows the implementation of the getModelById method. The function is send the DTO of GetSMSRequest and receive the DTO of GetSMSResponse, and then return DTO of ModelSpec.

```
public class DataWrapperClientImpl extends EnrimaClient implements DataWrapperClient {
@Override
   public ModelSpec getModelById(int idModel) {
     GetSMSRequest request = objectFactory.createGetSMSRequest();
     request.setIdModelSpec(idModel);
    GetSMSResponse response = (GetSMSResponse) getWebServiceTemplate()
                                                    .marshalSendAndReceive(request);
     return response.getModelSpec();
  }
}
```

Figure 5 the code snippet of the DataWrapperClientImpl.java

- App.java is the Java Main method which can be executed from the command line, It takes four arguments: action type (either mainset or entity), CSV file path, model id, model data id.
- SymbolicModelSpecificationHelper.java and XMLFormatter.java are the static helper classes which can format DTOs and XMl files.
- csv/CSVReader.java and csv/CSVParser.java provide fast access to CSV data. The Data Warpper developers can develop the XLS parser and local database Reader in the similar way.

- directory: wrapper/src/main/resources
  - mainset_*.csv, main set data examples.
  - demand.csv, entities values example.
  - log4j.properties, log level configuration file.
  - datawrapper-applicationContext.xml, the Spring application context configuration, it contains the setting of the marshaler, unmarshaler, DTO packages and the Web Services URL.

5 Compilation of the wrapper

5.1 Environment

- JDK1.6 or above
- Apache Maven 3.0.3 or above
- Git (optional)

5.2 Download the Kernel repository

$ git clone git://github.com/enrima-dev4/kernel.git

If the git is not available, then one can interactively zip the Kernel repository posted at https://github.com/enrima-dev4/kernel, and download the zipped repository.

5.3 Generate executable jar (enrimaDataWrapper.jar)

$cd kernel/data_wrapper
$mvn -Dmaven.test.skip=true clean install
$cd wrapper&&mvn -Dmaven.test.skip=true package
$cd target
$mv *one-jar.jar enrimaDataWrapper.jar

6 Install and run the Data Wrapper at a site

6.1 Environment

- JRE1.6 or above
- Internet connection

6.2 Install and Run the examples

- Copy the enrimaDataWrapper.jar to each site
- Upload mainset data example:
  $java -jar enrimaDataWrapper.jar mainset xx.csv modelId modelDataId
- Upload entity values example:
  $java -jar enrimaDataWrapper.jar entity xx.csv modelId modelDataId

7 Integrating the wrapper with the test-GUI

Add the following dependence to the core module of the test-GUI (test_gui/core/pom.xml)

```
<dependency>
     <groupId>at.ac.iiasa.ime.enrima</groupId>
     <artifactId>enrima-data-wrapper</artifactId>
     <version>1.0</version>
</dependency>
```