The developer guide to the test_gui

Hongtao Ren, Marek Makowski, IIASA

Version 0.2, February 25, 2012

1 Scope of this document

This guide aims at supporting the developers in using test_gui for the kernel testing, and in adapting the test_gui 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 test_gui
- Compilation of the test_gui
- Installation of the test_gui

2 Functionality of the test_gui

The test_gui is a simple web-based application developed for serving the following purposes:

- Illustrate how easy is to consume the WSs (Web-services) for a wide range of functionalities typically provided through a GUI
- Provide a template for other applications consuming WSs for data management
- Provide actual functionality of uploading model components: members of sets, parameters, and variables through the user friendly interface
- Provide the functionality of catching any exceptions and reporting to IIASA Jira system

Short characteristics of the test_gui:

- developed in Spring
- provides testing environment for the Kernel prototype
- seamless integration with the Jira client
- seamless integration with the data wrapper

Basic functionality of the test_gui:

- uses the SMS of the test model (id 71) described in the documentation directory
- gets the SMS
- uploads data from XML files

- uploads data from CSV file (through the data wrapper[1])
- uploads data (members of sets and values of parameters) from the DW
- supports interactive data handling (adding, modification, removing)
- uploads added/modified data to the DW
- supports creating analyses, specification of the corresponding preferences, calling solver, displaying solution [NOT fully functioning yet]

3 Overview the architecture

As a typical web application, the test_gui can be deployed into any web application server (Apache Tomcat, Jetty, etc). Figure 1 shows the WS-based interface between the Kernel and the test_gui.

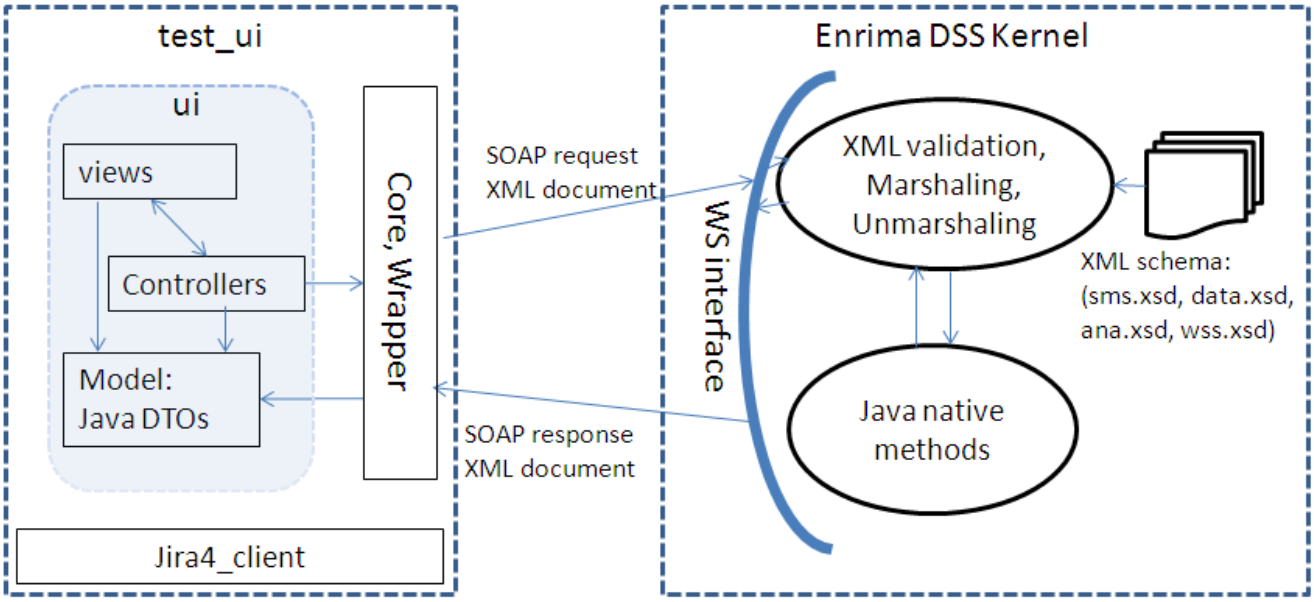

Figure 1 Kernel and test_gui interface

The test_gui is composed of the following three modules, each providing a specific functionality:

- Jira4_client: interface to the bug-tracking utility Jira
- core: integrate with the Data Wrapper; It contains all stubs for consuming WSs
- ui: provides web-based user friendly interface, it was designed and developed using the Spring Web model-view-controller (MVC) framework.[2]

Such a modular structure also demonstrates how easy it is to extend the functionality through adding the corresponding modules.

[1] Documented in a separate developer's guide

[2] http://static.springsource.org/spring/docs/3.0.x/reference/mvc.html

4 Files and directories in each module

4.1 Jira4_client module

- Jira4_client/pom.xml
  The Apache Maven (http://maven.apache.org/) project configuration file contains axistools-maven-plugin (http://mojo.codehaus.org/axistools-maven-plugin/) which can generate Java DTOs automatically based on a specification available in the WSDL file. Figure2 contains a snippet from the pom.xml (posted at the Kernel repository) file illustrating the plugin using the WSDL from jira4_client/src/main/resource/jirasoapservice-v2.wsdl and generating the DTOs into the package "at.ac.iiasa.ime.jira.soap"

```
<configuration>
      <sourceDirectory>${basedir}/src/main/resources </sourceDirectory>
      <wsdlFiles>
         <wsdlFile>jirasoapservice-v2.wsdl</wsdlFile>
      </wsdlFiles>
     <packageSpace>at.ac.iiasa.ime.jira.soap</packageSpace>
     <testCases>false</testCases>
     <serverSide>false</serverSide>
</configuration>
```

Figure2 code snippet of the jira4_client/pom.xml

- Directory: jira4_client/target/generated-source/axistools/wsdl2java
  The automatically generated Java DTOs corresponding to the XML schemas described in the jira4_client/src/main/resource/jirasoapservice-v2.wsdl, the Figure3 is the code snippet of at/ac/iiasa/ime/jira/soap/RemoteIssue.java is generated automatically based on the XML Data type of the remoteIssue described in the WSDL file.

```
package at.ac.iiasa.ime.jira.soap
 ...
public class RemoteIssue {
   private String assiginee;
   private String[] attachmentNames;
   private  Calendar created;
   private String project;
   private String status;
   private String description;
   ...
}
```

Figure3 code snippet of the RemoteIssue.java

- Directory: jira4_client/src/main/java
  Mainly contains Jira adapter which can convert user comments and Exceptions to the DTO of RemoteIssue and send the RemoteIssue to the IIASA Jira system (see the java class in the Kernel repository).

- jira4_client/src/main/resources/jirasoapservice-v2.wsdl

The WSDL file is downloaded from the IIASA configuration of the Jira system (URL: http://www.ime.iiasa.ac.at/jira/rpc/soap/jirasoapservice-v2?wsdl)

4.2 core module

- core/pom.xml
  It contains the project dependencies can be found within the <dependencies> </dependencies>. We can see from Figure 4, the core mainly depends on the Data Wrapper and the Spring Web Services (http://static.springsource.org/spring-ws/sites/2.0).

```
<dependency>
        <groupId>at.ac.iiasa.ime.enrima</groupId>
        <artifactId>enrima-client-domain</artifactId>
        <version>1.0</version>
</dependency>
<dependency>
        <groupId>at.ac.iiasa.ime.enrima</groupId>
        <artifactId>enrima-data-wrapper</artifactId>
        <version>1.0</version>
</dependency>
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.framework.version}</version>
</dependency>
```

Figure4 code snippet of the core/pom.xml

- directory: core/src/main/java/at/ac/iiasa/ime/enrima/client/ws/
  - ModelClient.java is the interface which declares all client stubs.

```
public interface ModelClient {

   public List<MemberDic> getSetMembers(GetSetMembersRequest getSetMembersRequest);
   public RemoveSetMembersResponse removeSetMembers(
                                   RemoveSetMembersRequest removeSetMembersRequest);
   public StoreSetMembersResponse saveSetMember(StoreSetMembersRequest request);
   public StoreMainSetResponse saveMainset(StoreMainSetRequest request);

   public GetEntityValuesResponse getEntityValues(
                                   GetEntityValuesRequest request);
   public StoreEntityValuesResponse saveEntityValues(
                                   StoreEntityValuesRequest request);
   public RemoveEntityValuesResponse removeEntityValues(
                                   RemoveEntityValuesRequest request);

   public DefineAnalysisResponse saveAnalysis(DefineAnalysisRequest request);
   public GetAnalysesResponse getAnalysesRequest(GetAnalysesRequest request);

   public DefinePreferenceResponse definePreferenceRequest(DefinePreferenceRequest request);
   public GetPreferenceResponse getPreferenceRequest(GetPreferenceRequest request);
   public GetSolutionResponse getSolutionRequest(GetSolutionRequest request);
   public StoreSolutionResponse storeSolutionRequest(StoreSolutionRequest request);

   public GetSolverStatusResponse getSolverStatus(GetSolverStatusRequest request);
   public UpdateSolverStatusResponse updateSolverStatus(UpdateSolverStatusRequest request);
}
```

Figure 5 the code snippet of the ModelClient.java

- ModelClientImpl.java is the implement of the interface ModelClient, the Figure 6 shows the implementation of the getEntityValues method. The function is send the DTO of GetEntityValuesRequest and receive the DTO of GetEntityValuesResponse.

```
@Override
public GetEntityValuesResponse getEntityValues(GetEntityValuesRequest request)
{
 GetEntityValuesResponse response =
(GetEntityValuesResponse)getWebServiceTemplate().marshalSendAndReceive(request);
 return response;
}
```

Figure 6 the code snippet of the ModelClientImpl.java

- Directory: core/src/test/java
  It contains sets of Junit tests to ensure the code in the core module works as intended. To run these unit tests:
  $cd core
  $mvn test
- core/src/main/resources/enrima-ws-client-applicationContext.xml
  The Spring application context configuration; it imports the configuration file of the Data Wrapper, and also declares the modelClient to be the interface to the DSS kernel.

4.3 Ui module

- ui/pom.xml
  It contains the project dependencies can be found within the <dependencies> </dependencies>. We can see from Figure 7, the core mainly depends on the core module and the jira4_client module.

```
<dependency>
        <groupId>at.ac.iiasa.ime.enrima</groupId>
        <artifactId>enrima-gui-core</artifactId>
        <version>1.0</version>
</dependency>
<dependency>
        <groupId>at.ac.iiasa.ime</groupId>
        <artifactId>jira4-client</artifactId>
        <version>2.0</version>
</dependency>
```

Figure7 code snippet of the ui/pom.xml

- ui/src/main/resources/application.properties
  The general configuration file, there are four blocks of the code in the file:
  - The first block is the settings of the application version, enrima.ver.type has two possible values d and p which corresponding the development version and public version. For the development version, all the exceptions will be caught and display in the web browser, for the public version, all the exceptions will be caught and sent to IIASA Jira system via the Jira4_client.
  - The second block is the jir4_client settings.

- The third block is the settings of the uploaded files path, before the installation of the test_gui to your local web application server; you should change the value of the uploaded_file_path accordingly.
- The fourth block is the preset of the model id, data id and model instance id.

```
enrima.ver =1.0
enrima.ver.type=d
#enrima.ver.type=p

contact.base.url=http://www.ime.iiasa.ac.at/contact/comment
app = enrima
jira.url=http://www.ime.iiasa.ac.at/jira/rpc/soap/jirasoapservice-v2
projectId=ENRIMA
componetId=10270
jiraUserName=comment_tool
password=xxxx
assignee=renh

uploaded_file_path=/home/hongtao/upload/
#download_file_path=/home/hongtao/download/examples.zip

modelId=71
dataId=23
modelInstanceId=6
```

Figure7 the application configuration

- Directory: ui/src/main/java

  It contains sets of Controllers that interpret the user input, call the core module, and return DTOs. The DTOs content is presented to the user by the View components. The following example shows the behaviors of the Controller of the AnalysisController.java:
  - receives the http request from the relative URL path: model/{modelId}/instance/{modelInstanceId}/analysis
  - interprets the request to the DTO of the GetAnalysesRequest
  - calls the core module, adds the returned DTOs of the List of Analysis to the Model
  - return the view 'analyses'

```
@RequestMapping(value = "/model/{modelId}/instance/{modelInstanceId}/analysis", method =
RequestMethod.GET)
public String populateAnalyses(
        @PathVariable("modelInstanceId") int modelInstanceId, Model model) {
                GetAnalysesRequest request =objectFactory.createGetAnalysesRequest();
                request.setIdModelInstance(modelInstanceId);
                   // call the core module
                GetAnalysesResponse response = modelClient.getAnalysesRequest(request);
                model.addAttribute("analyses", response.getAnalysis());
                  //return the view: analyses
                return "analyses";
        }
```

Figure7 code snippet of the AnalysisController.java

- Directory: ui/src/main/webapp

It contains sets of Views, for simplifying the development, the Apache Tile (http://tiles.apache.org/), JQuery (http://jquery.com/), JSTL (http://jstl.java.net/) are chosen as the view technology.

5 Compilation of the test_gui

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 Web application archive file (test_gui/ui/target/enrima-web.war)

$cd kernel/data_wrapper
$mvn -Dmaven.test.skip=true clean install
$cd ../test_gui
$ mvn -Dmaven.test.skip=true clean install

6 Install and run the test_gui

Copy the test_gui/ui/target/enrima-web.war to any web application server (Apache tomcat, Jetty, etc), If there is no existing web application in the environment, you can simply use Maven Tomcat plugin (http://tomcat.apache.org/maven-plugin-2.0/):

$cd test_gui/ui
$ mvn tomcat7:run

And then use the link http://localhost:8888/enrima to access the test_gui.