CERN
CH-1211 Geneva 23
Switzerland

the
A & B
Department

AB-BDI Software Section Documents

Project Name.
ROSALI

EDMS No.
-

Date: 2005-07-29

Technical Description

ROSALI Maintenance guide

Abstract

This document tries to put in place all ROSALI infrastructures to help new developers and system administrators in the tasks of maintenance, upgrade and new deployments.

Edited By:	Checked By:	Approved By:
Monica Moles	Xx	Xx

History of Change

Version	Date	Pages	Change Description
1.0	4/21/2005	All	Initial Submission
1.0	6/14/2005	5	Deployment diagram changed
1.0	6/29/2005	10	Section “Change action sequence structure” added.
1.0	7/29/2005	5,6	Changed location of the executables.

Table of Contents

Table of Figures

1. INTRODUCTION

The purpose of this document is to help new ROSALI developers and application administrators in the tasks of maintenance, upgrade and new developments. This document shall be used in conjunction with the following documents:

- ROSALI Coding guidelines.
- ROSALI Software Requirements Specification.
- ROSALI Design documentation.

2. ROSALI INFRASTRUCTURE AND LOCATION

The component diagram in Figure 1 represents the implementation perspective of the system. It gives developers a general view about the location of the software such as libraries, configuration files, etc.

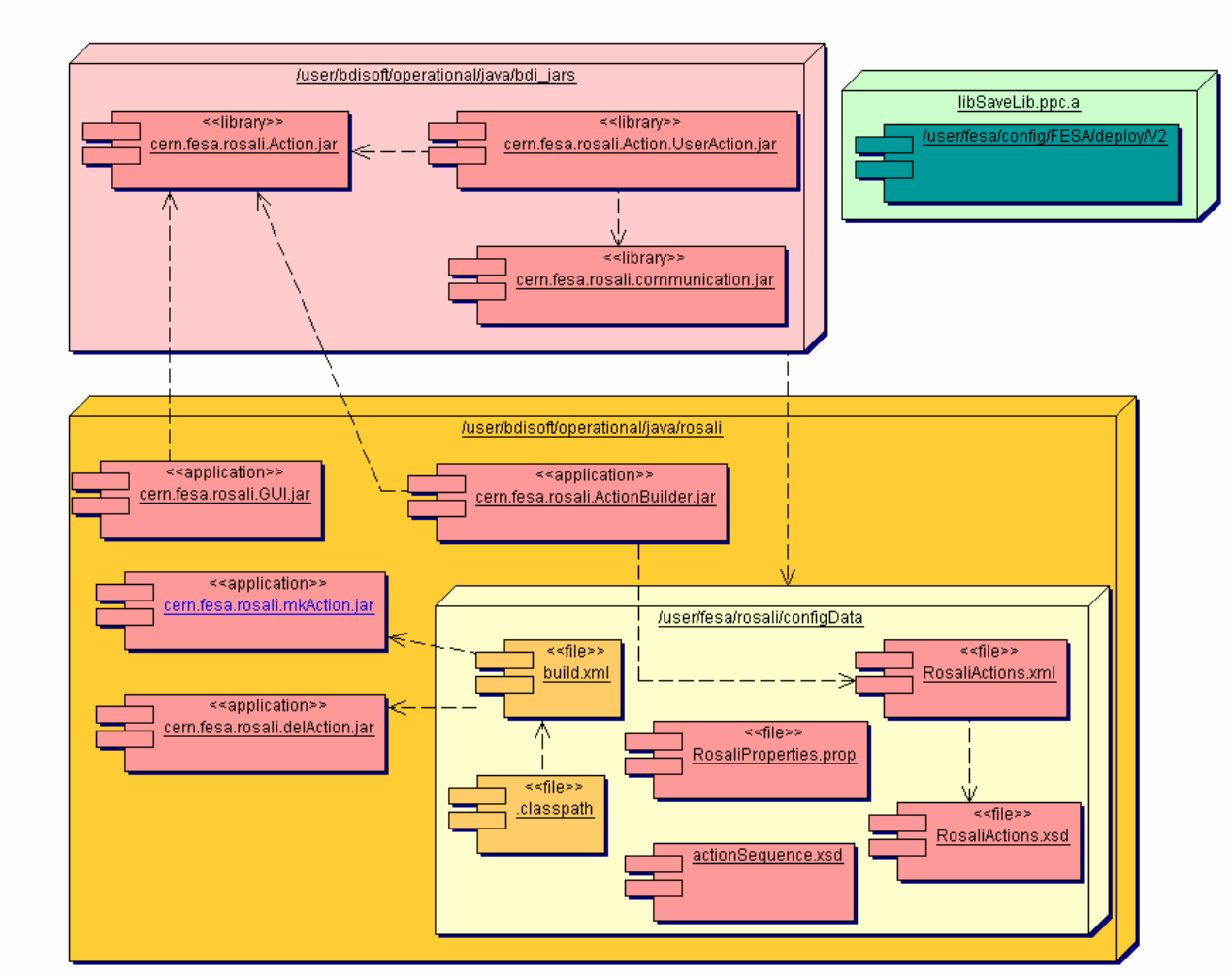

Figure 1 – Deployment diagram

The green box at Figure 1 represents the C library used by FESA instruments to generate their output in the ROSALI format. It is placed in `/user/fesa/config/FESA/deploy/V2`. To see in more detail its design and implementation, refer to “Savelib package decomposition

description" in the ROSALI design documentation. The user's manual can be found in the ROSALI web page under Documentation → Developer's corner → ROSALI Front-End library.

The pink box at Figure 1 contains all the libraries needed to run the ROSALI application. They are under `/user/bdisoft/operational/java/bdi_jars` containing the following modules:

- **`cern.fesa.rosali.Action.jar`** is a library that provides access to the Rosali action common features. It contains the implementation of the Action package defined in the Rosali Design Documentation.
- **`cern.fesa.rosali.Action.UserAction.jar`** contains all the `.class` files for the implementation of the users' actions.
- **`cern.fesa.rosali.communication.jar`** is a library that provides methods for creating and accessing files in Rosali format. It can be used by the users' defined actions or as a stand-alone jar file.

The yellow box at Figure 1 contains all the specific executables for the Rosali application located under `/user/bdisoft/operational/java/rosali`, like:

- **`cern.fesa.rosali.GUI.jar`** is the executable JAR file for the ROSALI GUI application. It contains the `.class` files for the implementation of the GUI and XmlParser packages defined in the Rosali Design Documentation.
- **`cern.fesa.rosali.ActionBuilder.jar`** is the executable JAR file for the ActionBuilder application. It contains the implementation of the ActionBuilder package defined in the Rosali Design Documentation.
- **`cern.fesa.rosali.mkAction.jar`** contains the application for creating a new Rosali action. This application checks if the action already exists, if not, it creates the action template in the working directory and the HTML page to write the action's user manual.
- **`cern.fesa.rosali.delAction.jar`** contains the application for deleting a Rosali action. It only deletes the action from the list of available Rosali actions, not deleting the source code from the user's working directory.

It also contains the configuration files under the `configData` directory:

- **`RosaliActions.xml`** XML file with all the available Rosali actions.
- **`RosaliActions.xsd`** XML schema defining the structure of the `RosaliActions.xml` file.
- The sequence of actions created by the user is saved in an XML file. **`actionSequence.xsd`** is the XML schema defining this structure.
- **`RosaliProperties.prop`** is used by the GUI application and sets the location of the XML files, the list of super-users and some other properties used by the application.
- **`build.xml`** Ant file used to create a new user action, compile it and deploy it.
- **`.classpath`** Eclipse project's classpath file defining the location of the external JAR files needed to compile the project.

The `standAlone` directory contains the executable for setting up the application. It also contains the script used to generate this setup file. The program used for that is Inno Setup 5 and can be downloaded from http://www.jrsoftware.org/isinfo.php .

3. MODIFYING THE SOURCE CODE

3.1. PRELIMINARY STEPS

All source code is under CVS on `bdidev1` under the module `ROSALI`. At Figure 2 you can see a snapshot of the modules under the ROSALI module.

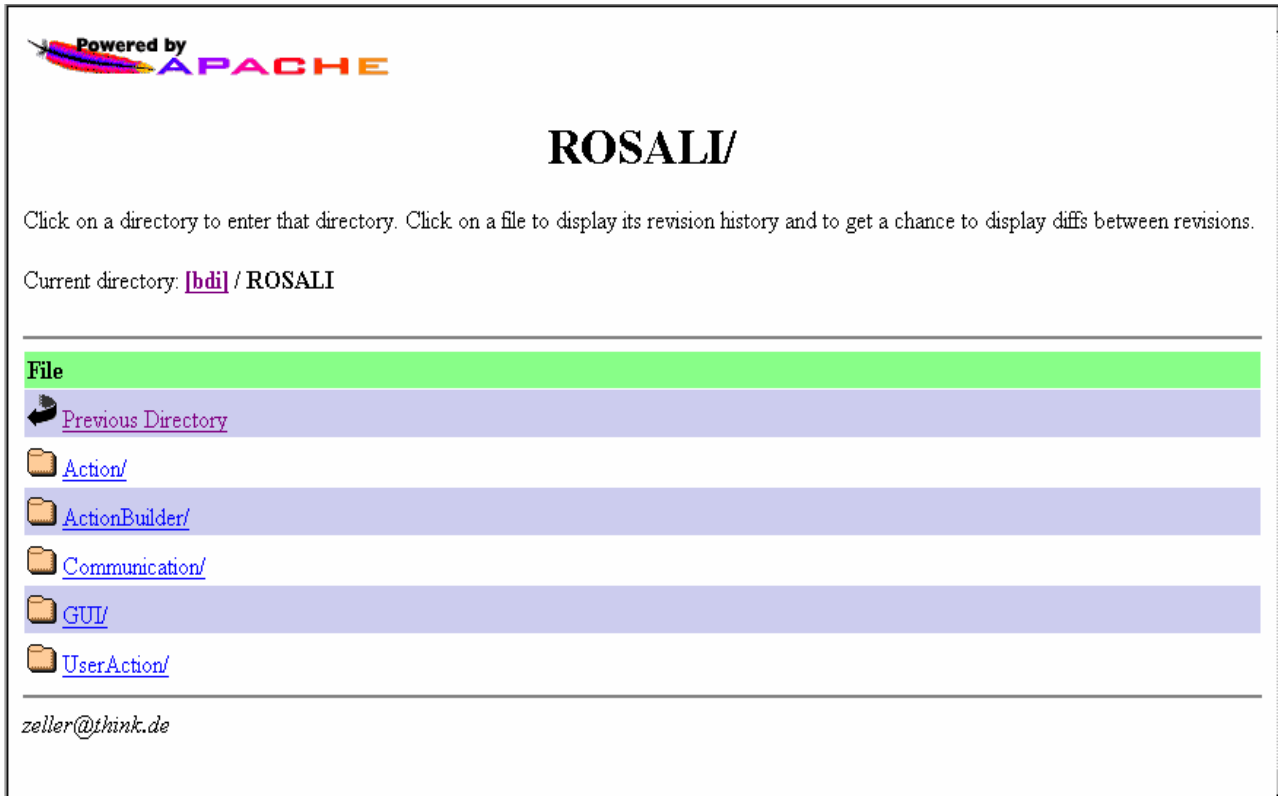

Figure 2 – View of the ROSALI module in CVS

The tool recommended to modify the code is Eclipse. First of all you have to configure your CVS Repository under Eclipse to check out the code. If you haven't done it yet, please follow the steps:

1. Click on Window → Show view → Other... and then select "CVS Repositories".
2. Right-click in the new panel and select New → Repository Location. Window showed in Figure 3 will appear. Enter the info like in the figure, then press Finish:

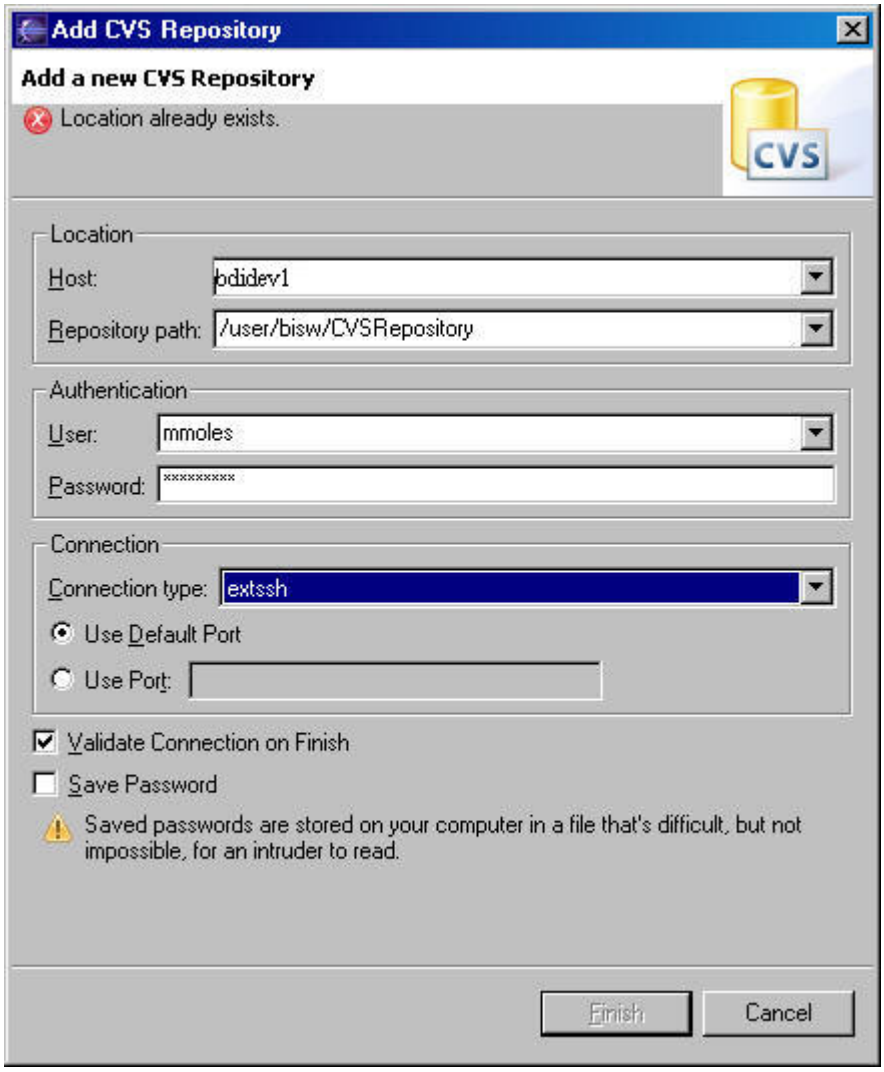

Figure 3 – Configure CVS repository under Eclipse

3.2. CHECKING OUT A PROJECT

Then, you should check out the module you want to modify from the CVS server. To do so:

1. Right-click on the Package Explorer tab located on the left side of the window.
2. Select “Import” from the pop-up menu. Figure 4 will appear.

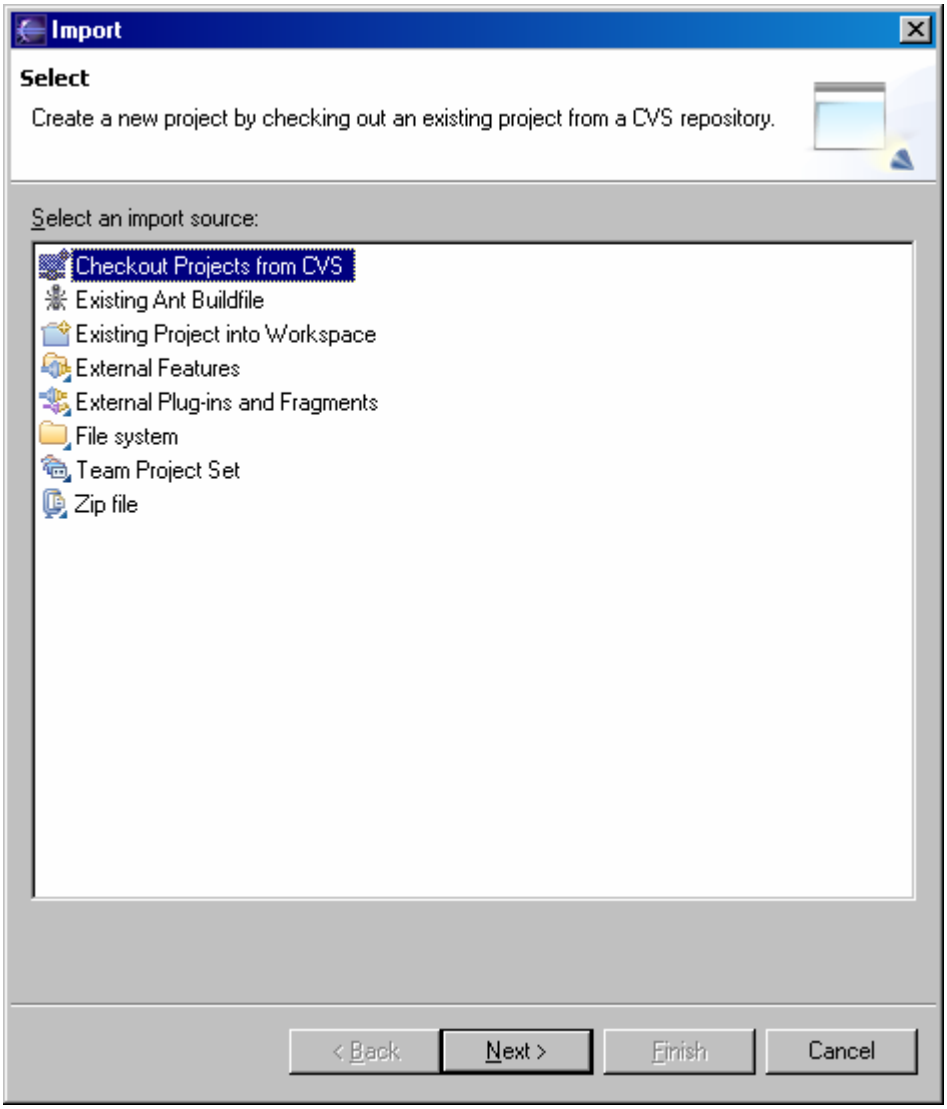

Figure 4 – Import project from CVS step 1

3. Select “Checkout Projects from CVS” and press “Next”.
4. In the next window, select the option “Use existing repository location” and press “Next”.
5. In the next window titled “Select module”, select the option “Use an existing module”. Then Eclipse will connect to the CVS server (maybe you need to enter your password) and it will list the modules existing on the server. All the ROSALI packages are under the ROSALI module. Select the module you want to check out and press “Next”.
6. Select extract from HEAD and press Finish. It will create a new project on the left window.

3.3. MODIFY THE CODE

After checking out the project from CVS, you should set the environment for compiling and testing it. All modules come with an Ant file with a target called “`setup_project`”. Click on it and then refresh the project (select project, right-click and then “Refresh”).

Now, your project is ready for any change!!

3.4. DEPLOYMENT OF THE NEW CODE

After changing the code and testing it in the Eclipse framework, it's time to deploy the JAR file in either the test or production environment. As said before, all modules come with an Ant file with a target called "deploy_development" and another one called "deploy_operational". Select the one you need. It will create the Jar file and move it to the right place.

Then you should commit the changes in the CVS server.

3.5. CHANGE THE ACTION SEQUENCE STRUCTURE

If the developer needs to add a new field to the action, meaning to change the ActionTemplate.java file, this change should be also reflected in the action sequence schema.

When parsing a file with a sequence of actions, the SequenceParser class reads the structure the sequence should have. This structure is defined in the actionSequenceSchemaLoc property (see Figure 6 – RosaliProperties file). Once the structure of the sequence is loaded, the SequenceParser class reads the XML file containing the sequence. If it doesn't follow the structure of the schema, the file is not well-formed. So, if the developer adds a new field to the action, he/she should add it also to the sequence schema to read/create the file properly. For instance, if the developer needs to add a new type of field to the action namely checkbox, he/she should add it in the ActionTemplate.java file and in the actionSequence.xsd file. Currently, it has the following structure:

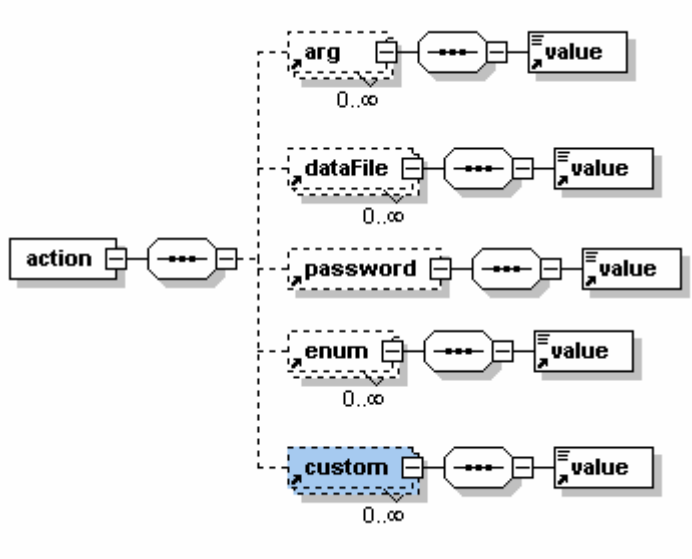

Figure 5 – XML Sequence schema

4. CHANGE ROSALI CONFIGURATION PARAMETERS

The configuration parameters for the ROSALI application are defined in the **RosaliProperties.prop** file under /user/fesa/rosali/configData. This is the current content of the file:

```
#Rosali Properties
defaultSettingFile=RosaliDefault.prop
actionSequenceSchemaLoc=rosali/configData/actionSequence.xsd
RosaliActionsSchemaLoc=rosali/configData/RosaliActions.xsd
RosaliActionsXML=rosali/configData/RosaliActions.xml
```

```
RosaliWebPage=http://project-rosali.web.cern.ch/project-rosali/ActionsDoc
RosaliSuperUsers=mmoles,sjackson,sbartped,sam,anag,ljensen,karlsson,alokhovi
```

Figure 6 – RosaliProperties file

The user's preferences such as the working and input directories are saved in a file whose name is defined in the property **defaultSettingFile**. If the user is working under the Windows environment, the file will be created in `c:/Documents and Settings/username`; otherwise under `/user/username`.

The **actionSequenceSchemaLoc** property sets the location of the XML schema file defining the structure of the sequences of actions.

The **RosaliActionsSchemaLoc** property sets the locations of the XML schema file defining the structure of the file with the Rosali actions.

The **RosaliActionsXML** property sets the location of the XML file with the list of available actions.

The **RosaliWebPage** property sets the URL where the actions' documentation can be found.

The users listed in the **RosaliSuperUsers** property have special permissions and can see every Rosali action.

5. ROSALI ACTIONS

The procedure to create a new action to use it in the ROSALI GUI is defined in the ROSALI web page under the section Documentation → Action's developers' corner → How to build a new action.

5.1. THE ANT FILE

For this guide, it requires a special attention the Ant file for creating new actions, compile and deploy them. When the user creates a ROSALI action for the first time, he creates a new project in Eclipse importing a ZIP file which contains a `build.xml` file and the `.classpath`. The `build.xml` has the following targets:

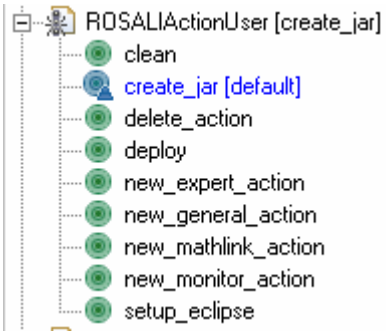

Figure 7 – Targets in the build.xml for creating a new ROSALI action

The **setup_eclipse** target should be executed each time the user starts a project or when an external jar file has changed. First of all, it copies the latest version of the `build.xml` and `.classpath` in your project directory. These files are taken from `/user/fesa/rosali/configData` like shows Figure 1. It also copies the external jar files needed to compile your project in the 'lib' directory.

The **new_general_action** target creates the Java code source file for the new action based on the general action template. It calls the mkAction application (refer section 5.3 "mkAction application"). The name of the action to create is set in the build.xml file. Change the name there.

The **create_jar** target compiles your project and creates the jar file that can be used to test your new action inside Eclipse. Once you are ready to deploy it, use the target **deploy** for it.

5.2. ACTION TEMPLATE FILES

The targets that create a new action make a copy of a template file changing the action's name and the file's name. At this moment, there are four templates files placed on /user/fesa/rosali:

- ActionAnonymous.java → Template file for a general action.
- ExpertAnonymous.java → Template file for an expert action.
- MathLinkAnonymous.java → Template file for an action that uses Mathematica.
- MonitorAnonymous.java → Template file for an action with monitoring features.

The application administrator can add as many templates as he considers. The only thing he/she needs to do is to create a template file following the remarks listed below and to add a new target in to the Ant file.

All the templates files must include:

- the package name cern.fesa.rosali.Action.UserAction
- the class name MUST be 'Anonymous' and extend ActionTemplate.
- the value for the name attribute in the constructor MUST be "Anonymous".
- The template should provide the methods verify, execute and restore, as well a brief description about how to use them.

Example of an action template.

```
// package should always be this
package cern.fesa.rosali.Action.UserAction;


import cern.fesa.rosali.Action.*;
import cern.fesa.rosali.communication.RosaliObjException;

/**
 * All actions defined by a developer will be a copy of
 * ANONYMOUS. That class defines the basis methods
 * a developer needs to implement. All the common behavior
 * will be managed by the ActionTemplate class.
 * @author Monica Moles
 * @since August 20th, 2004
 */
public class Anonymous extends ActionTemplate {
    /**
     * Constructor. There the developer will specify
     * the attributes his class need to use.
     */
    public Anonymous() {
```

```
    /**** PLEASE NOT REMOVE ****/
    this.name = "Anonymous";
    /**** END ATTRIBUTES ****/


    // Developer contact details for free!!
    this.developer = new DeveloperAnonymous();


    /**
     * Create viewer. You have 2 ways to create it:
     *
     * - new ActionViewer();
     *   it creates a panel containing the 2D, 3D and ASCII viewer
     *
     * - new ActionViewer(CustomPanel p)
     *   this constructor also adds a custom panel to the tab.
     *   The custom panel will be implemented by the developer and MUST
     *   extend 'CustomPanel'. By default it contains a single button.
     *
     * - You can always add the custom panel with
     *   viewer.addCustomPanel(customPanel);
     **/
    this.viewer = new ActionViewer();
  }

  /**
   * This method is called when an argument is changed by the User in the
   * fields panel.
   * If the developer wants to check the values entered before applying
   * them, he/she has to provide the code here!!!
   * - 'userFieldName' is the name defined by the user for the field and
   * - 'value' is the new value the user is trying to set. That value is
   *   not yet in this.argX or this.dataFileX.
   *
   * The user only has to verify the value, without setting it to the
   * attribute. If something goes wrong in the 'verify' method, the
   * developer must throw an exception (throw new Exception("message")) and
   * the application will show the message in a dialog. If there is no
   * exception, then the engine will apply the new value to the field.
   **/
  public void verify(String userFieldName, String value) throws Exception{


  }


  /**
   * This method is called when the action has to be executed.
   * You should provide your code here!!
   *
   * You can access to the viewers and preferences with:
   * - Add text in the Ascii viewer -->
   *                          this.viewer.addASCIIPanelText("Testing!!!!");
   * - Set text in the Ascii viewer -->
   *                          this.viewer.setASCIIPanelText("Testing!!!!");
   * - Access to the 2D viewer      --> this.viewer.view2D
   * - Access to the 3D viewer      --> this.viewer.view3D
   * - Access to the custom panel   --> this.viewer.viewCustom
   * - Get the default input directory -->
```

```
     *                                  this.preferences.getInputDirectory();
     * - Get the default Mathematica dir -->
     *                               this.preferences.getMathematicaDirectory();
     * - Get the default working dir     -->
     *                               this.preferences.getWorkingDirectory();
     * - Switch to a viewer tab -->
     *                     this.viewer.switchToPanel(ViewManager.ASCII_PANEL);
     * @throws RosaliObjException
     */
    public void execute() {

      this.viewer.addASCIIPanelText("Testing!!!!");
    }


    /**
     * This method is called when the user stops the action's execution
     * or when the tool gets an error during the execution. If the user
     * wants to restore some values in the action's arguments, his/her
     * code should come here!
     */
     public void restore() {

     }
}
```

Figure 8 – Action template file

5.3. MKACTION APPLICATION

The **mkAction** application is used in the Ant file for creating a new action. If you want to modify the code, it is in CVS under the module Admin. Figure 9 shows you the activity diagram with the tasks performed by the program.

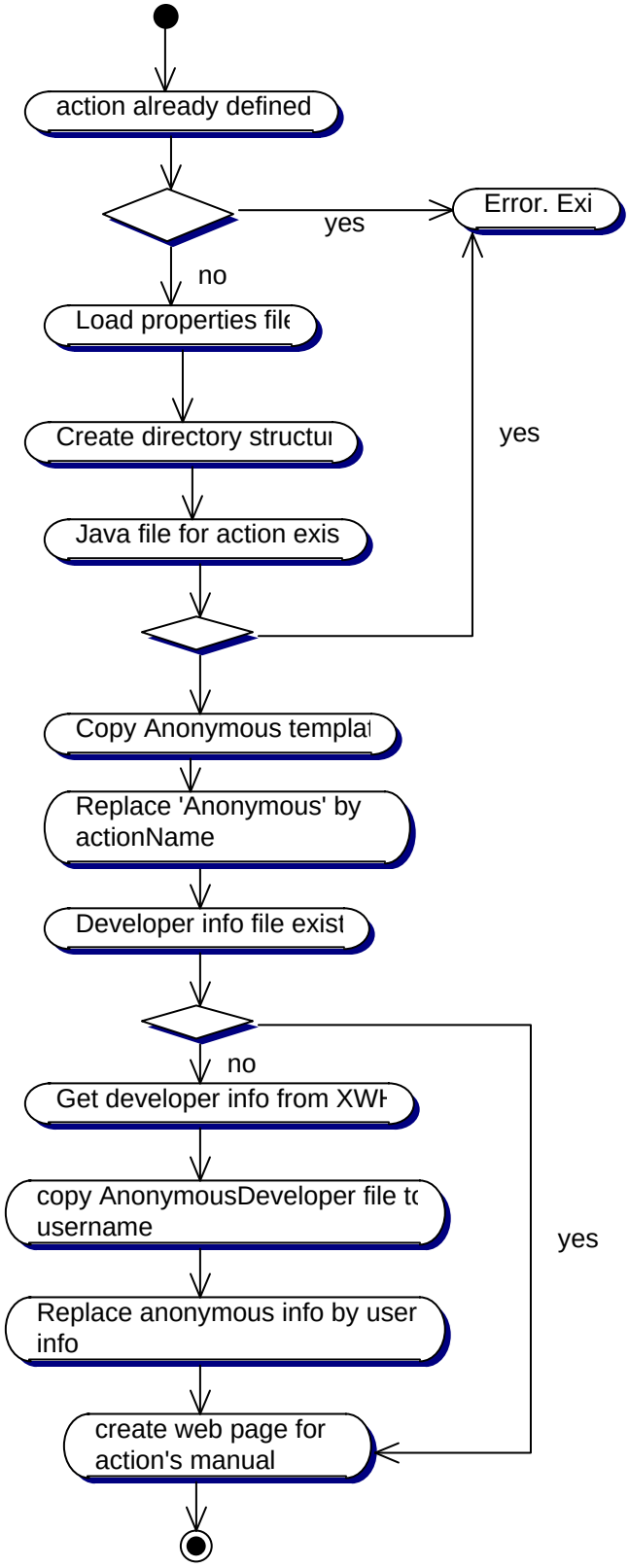

Figure 9 – Activity diagram for mkAction

The application reads the configuration parameters from the `mkActionProperties.prop` file located at `/user/fesa/rosali/configData`. This file is listed below and defines:

- The path of the package to be created (**userPackage**).
- The Java template file containing the developer's personal information (**developerFile**).
- The web server name for the Rosali web page (**webServer**).
- The project site name
- The directory where the actions' user manuals will be placed on the Rosali web page (**documentationPath**).
- The name of the web page used as template (**templateWebPage**).

```
#mkAction Properties
userPackage=/cern/fesa/rosali/Action/UserAction
developerFile=DeveloperAnonymous.java
webServer=webh04.cern.ch
projectWebSite=project-rosali
documentationPath=ActionsDoc
templateWebPage=Anonymous.htm
```

Figure 10 – mkActionProperties file

5.4. DELACTION APPLICATION

The **delAction** application is used in the Ant file to delete an action from the list of available Rosali actions. It doesn't delete the java source files from the user's directory. If you want to modify the code, it is in CVS under the module Admin.

6. CONFIGURE THE APPLICATION LAUNCHER

The ROSALI GUI and the ActionBuilder applications are launched from the Application launcher. The resources needed to launch them are already configured, but if you need to modify them.

1. Open the application launcher tool from the section web page.
2. Select the "Edit configuration" tab and open "BDI Generic applications".
3. Select the application you want to modify, i.e. ROSALIGUI and open jnlp → resources. If the location of the JAR files has changed, you should modify them here.

Figure 11 – Application launcher for ROSALIGUI

Note that both applications extend `ExpertGUI` application inheriting all its resources. Due to some version incompatibilities with the XML packages, the jar file `xmlparserv2` must be the first one on the list in order to work properly.

7. CHANGING THE DEPLOYMENT DIRECTORY

If for any reason, you need to change the location of the deployment directory, here there is a list of things you should do. Note that you MUST keep the default directory structure where the `RosaliPoperties.prop` and `mkActionProperties.prop` must be under `rosali/configData`.

1. First of all, change the JAR files to the new directory.
2. Change the value of the property `OPER_ROSALI_DIR` to the new location in every `build.xml` file.
3. Change the value of the property `ROSALI_JARS_DIR` to the new location in the UserAction `build.xml` file.
4. Update the location of the resources in the application launcher (see section 6 “Configure the application launcher”).