Project Acronym:	**OPTIMIS**
Project Title:	**Optimized Infrastructure Services**
Project Number:	**257115**
Instrument:	**Integrated Project**
Thematic Priority:	**ICT-2009.1.2 – Internet of Services, Software and Virtualisation**

Integrated Development Environment User Guide

Activity 2: *Service Construction*

WP 2.1: *Programming Model and IDE*

Due Date:		M34
Submission Date:		31/04/2012
Start Date of Project:		01/06/2010
Duration of Project:		36 months
Organisation Responsible for the Deliverable:		BSC
Version:		1.0
Status		Final
Author(s):	Jorge Ejarque	BSC

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	10/04/2012	First version	Jorge Ejarque (BSC)
0.2	20/09/2012	Updated Version	Jorge Ejarque (BSC)
0.3	26/03/2013	Updated Version	Jorge Ejarque (BSC)
1	07/06/2013	Final version	Malena Donato (ATOS)

Table of Contents

1 INTRODUCTION 6

1.1 GLOSSARY OF ACRONYMS 6

2 INTEGRATED DEVELOPMENT ENVIRONMENT USER GUIDE 7

2.1 RELEASE INFORMATION 7
2.2 INTRODUCTION 7
2.3 FUNCTIONALITIES 7
2.3.1 Create a new Project 7
2.3.2 Create a new Service Class 7
2.3.3 Create a Orchestration Element in a Service Class 7
2.3.4 Create a Core Element Service Class 8
2.3.5 Add Elasticity to a Method Core Element 8
2.3.6 Add Elasticity to an Orchestration Element 8
2.3.7 Add Processing, Storage and Licensing Constraints to Elements 8
2.3.8 Import Application from the war and jar packages 8
2.3.9 Create service packages 9
2.3.10 Deploy in Localhost 9
2.3.11 Deploy a service in the OPTIMIS infrastructure 9
2.4 KNOWN LIMITATIONS 9
2.5 GETTING STARTED 10
2.5.1 Using the Software 10
2.5.2 Testing the Software 15
2.5.3 Configuration 15
2.6 OTHER INFORMATION 16
2.6.1 Source Code Information 16
2.6.2 Contributors 16

3 REFERENCES 17

Table of Figures

Figure 1. OPTIMIS IDE Service Editor 10
Figure 2. Service Class wizard 11
Figure 3. Orchestration Element Wizard 12
Figure 4. Core Element Wizard 13
Figure 5. Orchestration Element implementation 14
Figure 6. Building and Deployment Tab 15

1 Introduction

This document includes the user guide for the software component Integrated Development Environment (IDE) The following sections describes how to use the component interface in order to get the expected functionality.

1.1 Glossary of Acronyms

Table 1 - Acronyms table

Acronym	Definition
D	Deliverable
WP	Work Package
IDE	Integrated Development Environment

2 Integrated Development Environment User Guide

2.1 Release information

Table 2 - release component information

Component Name	Release Number	Release Date
Integrated Development Environment	1.0.0	30/9/2012

2.2 Introduction

The IDE plugin extends the Eclipse [1] capabilities to implement Java applications with a set of menus actions and wizards to facilitate the implementation of services with the OPTIMIS programming model. In addition to the implementation capabilities, the IDE also provides an interface for facilitating and automating the creation of the service packages for the deployment in the OPTIMIS infrastructure.

More details about the architecture of the OPTIMIS IDE can be found in the Section 14 of the Detailed Design document [2].

2.3 Functionalities

Next paragraphs describe how to achieve the main functionalities provided by the IDE component

2.3.1 Create a new Project

A user can start the wizards for creating a new OPTIMIS service project, using the standard eclipse menu “File-> New… -> Project” or the “Create Project” action in the “OPTIMIS-> Implementation” menu. This wizard contains some fields to set the name of the project and the main package. There is also a field to select the location where the Programming model runtime is installed.

Once you have filled these three fields, the user can either define other Java project settings in the second page or finalize the wizard. At the end of the wizard, the IDE automatically creates the source folder with the main package and a core elements sub-package. It also creates some metadata files to store the project configuration and open a Service editor where the user can visualize and manage the implementation of the service.

2.3.2 Create a new Service Class

Once a project is created a user can create a service class clicking “New…” in the Service class section of the Service Editor or selecting the “Create Service Class” action in the “OPTIMIS-> Implementation” menu. This wizard contains a field to set the class name and it optionally allows the user to change the package and source folder of the service class. Once the wizard is finalized a new service class and its corresponding Core Element Interface are created and automatically the new service class automatically appears in the Service Editor.

2.3.3 Create a Orchestration Element in a Service Class

Once a Service Class is created a user can create an Orchestration Element, selecting the Service Class in the combo box and clicking “New…” in the Orchestration Element Section of the Service Editor. The Orchestration element wizard contains some fields to set the element name, it input parameters return values, constraints and if it is going to be part of the Web

Service interface. Once the wizard is finalized a new method is created in the service class with the corresponding parameters and annotations.

2.3.4 Create a Core Element Service Class

Once a Service Class is created a user can create an new Core Element, selecting the Service Class in the combo box and clicking "New…" in the Core Element Section of the Service Editor. In the first page of the New Core Element wizard, the user can select the way to add a new Core Element from the following options.

- **Method Core Element from scratch**, where the user provides all the parameters, return types and constraints.
- **Method Core Element from an existing class**, where the user selects a method from a existing class, adding only the constraints.
- **Method Core Element from Executable,** where the user select the command or binary and the arguments and which are going to appear in the Core Element interface.
- **Service Core Element from WSDL**, where the users select a method from a deployed web service.
- **Service Core Element from a war package,** where the users select package location and the service invocation of a web service of this package**.**

Once the wizard is finalized a new method is created in the service class interface with the corresponding parameters and annotations. In the case of a Method Core Element from scratch or executable a new class and method is created within the Core Element sub-package.

2.3.5 Add Elasticity to a Method Core Element

Once a Method Core Element is created it can be selected in the Core Element Section of the Service Editor. The description of this Core Element will be printed in the Core Element Description section. This section contains a Elasticity description where the user can specify the maximum and minimum number of Core Element will be executed.

2.3.6 Add Elasticity to an Orchestration Element

Once an Orchestration Element is created it can be selected in the Orchestration Element Section of the Service Editor. The description of this Orchestration Element will be printed in the Orchestration Element Description section. This section contains an Elasticity description where the user can specify the maximum and minimum number of Elements will be executed.

2.3.7 Add Processing, Storage and Licensing Constraints to Elements

Once an Element is created it can be selected in the Element Section of the Service Editor. The description of this Element will be printed in the Orchestration Element Description section. In this section you will see a Constraints table and a set of buttons to Add, Modify and delete constraints. Click Add and select the type of constraint and set the value of the constraint. You can set Processing constraints like the number of cores, architecture or memory. The requirements of the scratch, shared and encrypted storage required by the elements as well as the license tokens if required.

2.3.8 Import Application from the war and jar packages

Once a project is created a user can import an application from existing packages by importing services class and defining orchestration and core elements from the classes and libraries contained in those packages. To import the service class click "Import…" in the Service class section of the Service Editor. Then, select the location of the package. Once the package is selected it will be extracted in the project's "imported" folder. Select the orchestration class by

selecting a library or the class folder contained in the package. After that, a new Service class will appear in the Service Editor.

Afterwards, select the orchestration elements by clicking *New...* in the Orchestration Element Section and selecting one of the methods of the imported class.

The core elements can be selected by clicking New… in the Core element section and follow the instruction of the Core Element Creation wizard for existing classes.

2.3.9 Create service packages

Once the implementation phase is finalized, the different Orchestration and Core Elements defined for a service can be grouped in packages using the Building and Deployment tab of the Service Editor. In the manual mode, the user defines which Orchestration and Core element will be included in each package. In the Automatic mode, the IDE will generate automatically this packages according to incompatible hardware constraints keeping the minimum number of packages. Once they have been defined the Orchestration and Core Element Packages, they are generated when clicking the “Generate” button.

2.3.10 Deploy in Localhost

Before deploying the service in a Cloud, the IDE offers the possibility of testing the service in the user’s machine. For doing this test the user only has to set the Deployment Type to “Localhost”, select the folder where the Apache Tomcat [3] is installed, select a folder to deploy the core elements and click the “Deploy” button. Then, the service will be deployed in the Tomcat’s container and its execution will be performed by means of the programming model runtime in order to test that the Orchestration Elements are executed as expected.

2.3.11 Deploy a service in the OPTIMIS infrastructure

To deploy an implemented service in the OPTIMIS infrastructure, the user has to select the OPTIMIS Cloud deployment option in the Building and Deployment Tab of the Service Editor. After selecting this type different subsection will appear in the Deployment Options, where the user can do the following actions:

Create Service Images, by defining the Image Creation Service location in the Image Creation subsection and clicking “Create Images”.

Generate License Tokens, by defining the License Manager server and the client configuration (user certificates location, etc) and clicking Generate for each of the license tokens specified in the Element constraints.

Define TREC parameters, by opening the different TREC sub-sections, creating or modifying current parameters and clicking “Save”.

Define Affinity and Anti-Affinity Rules, by defining the affinity or anti-affinity constraints between the different components and the different instances of the same component. Define Security and Data Protection, by selecting the Property rights for the data generated by the service and other legal requirements for delegating processing data to cloud providers, and other security properties.

Deploy the service, by defining the Deployment Service location in the Service Deployment sub-section and clicking “Deploy”

2.4 Known limitations

No limitation currently found

2.5 Getting Started

A Service developed by the OPTIMIS Programming Model is composed by Orchestration and Core Elements. A Core Element is a piece of code which either is repeated several times in the service code and potentially in parallel or is performing a computation which requires a lot of resources or both. An Orchestration Element is the code which implements the service functionality invoking several defined Core Elements.

2.5.1 Using the Software

If the IDE plug-in has been successfully installed an OPTIMIS menu with different actions should appear in the Menu bar and a set of OPTIMIS wizards should appear in the "File->New…" sub menu when the Eclipse platform is initiated. There is also a Service Editor () which will be available when a new project is created. You can use these actions, wizards and editor to easily create a new service with different Orchestration and Core Elements in ten steps.

Step 1.

Create a new OPTIMIS project with the New OPTIMIS project wizard located in "File ->New-> Project". Select the name of the project, the name of the main packages and the location where the OPTIMIS Programming Model is installed in your machine and click "Finish". As result of this wizard a new project and Service Editor will be opened as shown in Figure 1.

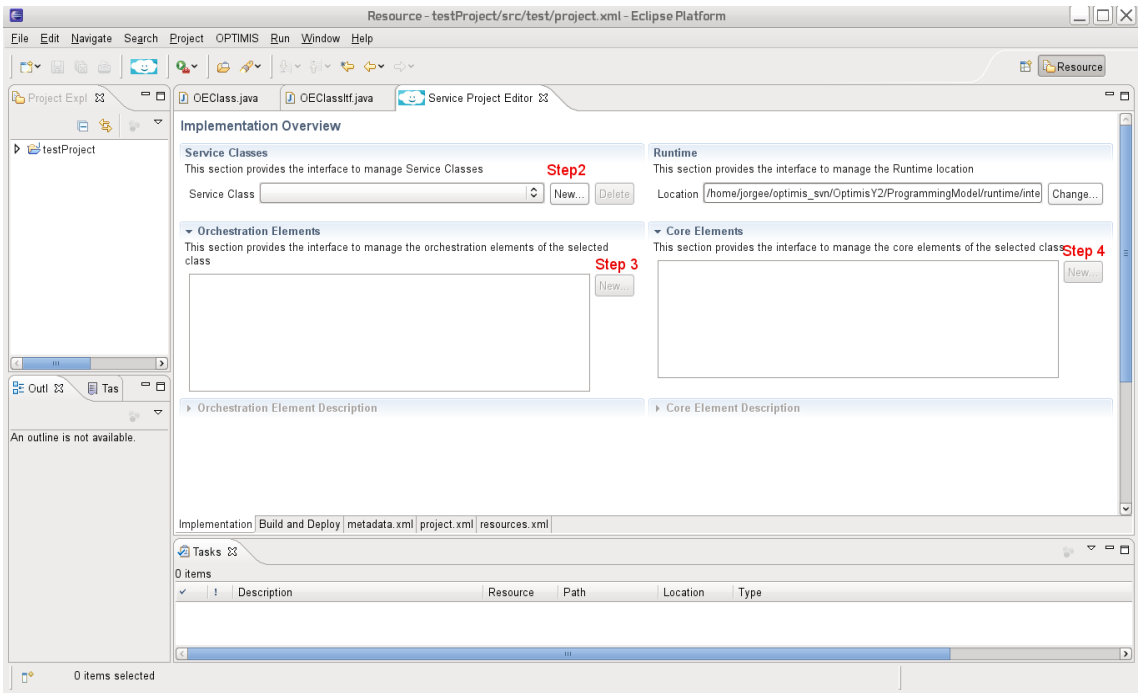

Figure 1. OPTIMIS IDE Service Editor

Step 2.

Create a new Service Class using the Implementation Tab of the Service Editor. A Service Class wizard (such as Figure 2) will be opened by clicking the "New…" button in the Service Classes

section (Figure 1. Step 2). After fulfilling the Service class name, a new class and a Core Element interface will be created in the main package of the project.

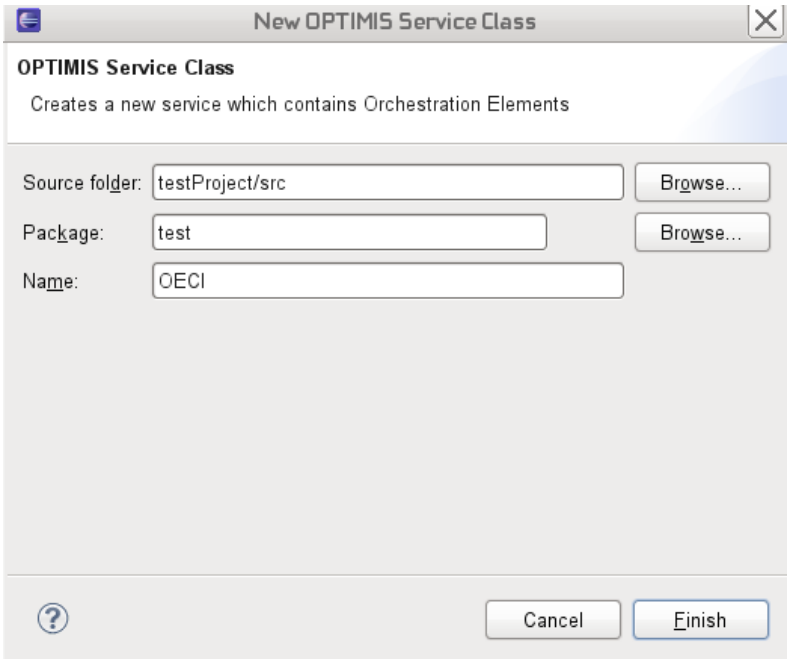

Figure 2. Service Class wizard

Step 3.

After the creation of the Service Class, you can include an Orchestration Element in this class where a workflow with different Core Element invocations will be programmed. You can create an Orchestration Element by using the Orchestration Element Wizard (Figure 3) opened when clicking the "New..." button of the Orchestration Element section located in the Implementation Tab of the Service Editor (Figure 1 Step 3)

Figure 3. Orchestration Element Wizard

Step 4.

The Core Elements invoked from the Orchestration Elements of a Service class must be defined in the Core Element Interface created when adding a Service Class. To do it the OPTIMIS ide offers a Core Element wizard which is open by clicking the "New…" button of the Core Element section located in the Implementation Tab of the Service Editor (Figure 1 Step 4). You currently have 3 options to create a new Core Element: From scratch, creating a new method in a Core Element class, from an existing JAR package or from an existing Web Service described by its WSDL file.

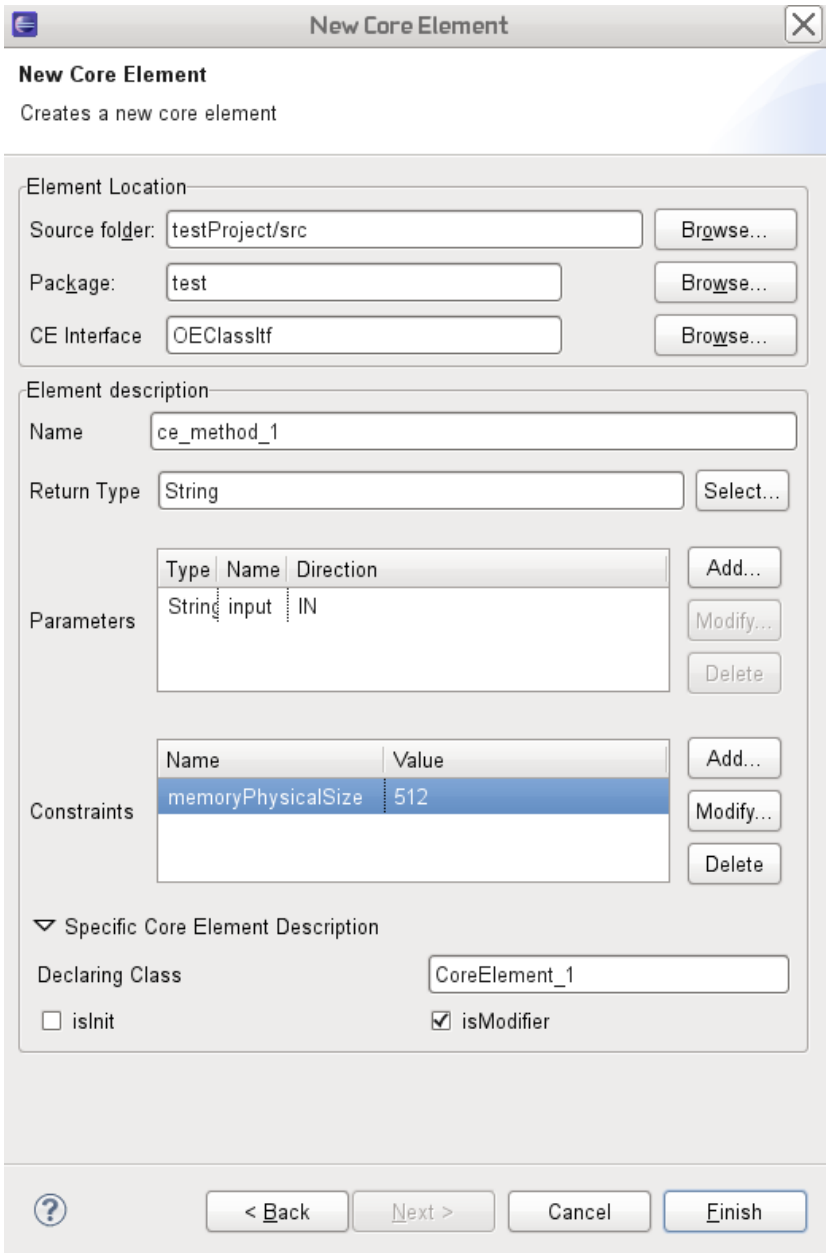

Figure 4. Core Element Wizard

Step 5.

Once a Core Element has been defined to the Core Element interface, the developer can implement its invocation inside the Orchestration Element code (as depicted in . Repeat step 4 and 5 to implement the service functionality as a composition Core Elements invocations in a sequential fashion.

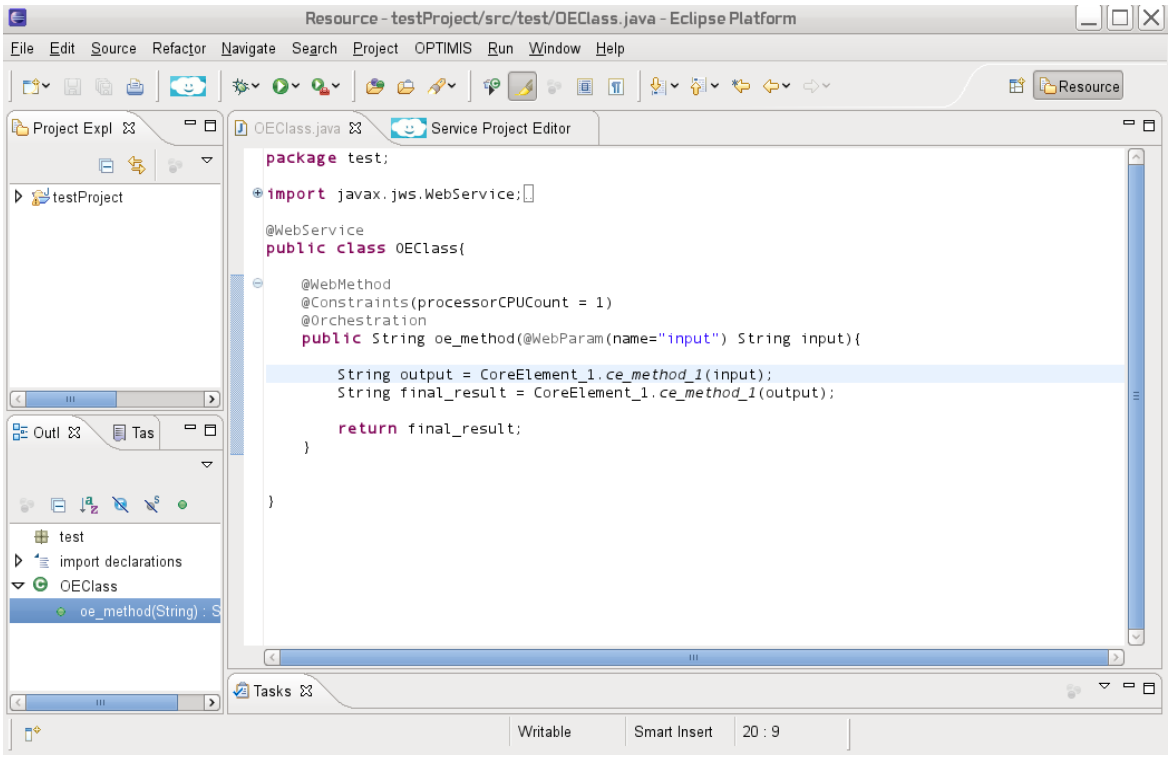

Figure 5. Orchestration Element implementation

Step 6.

Once you have implemented the service, move to the Building and Deployment Tab (Figure 6) of the Service Editor. Define the service packages by grouping the Core Elements with similar requirements or functionality by clicking “New...” button of the Service Packages section (Figure 6 Step 6). Once the Service Developer has defined the packages for all the Core Elements, click the “Generate” button to generate the defined packages (Figure 6 Step 6b). As result of the package generation process a set of war, jar and zip files will be generated in the “output” folder of the OPTIMIS Service project.

Step 7.

Check the correct behavior with a local deployment selecting the “Localhost” deployment option (Figure 6 Step 7) will deploy the Orchestration in a Web Application Server (such as Tomcat) and Core Elements in a selected folder of your machine. Once the developer has checked the correct behavior select the “OPTIMIS cloud” deployment option.

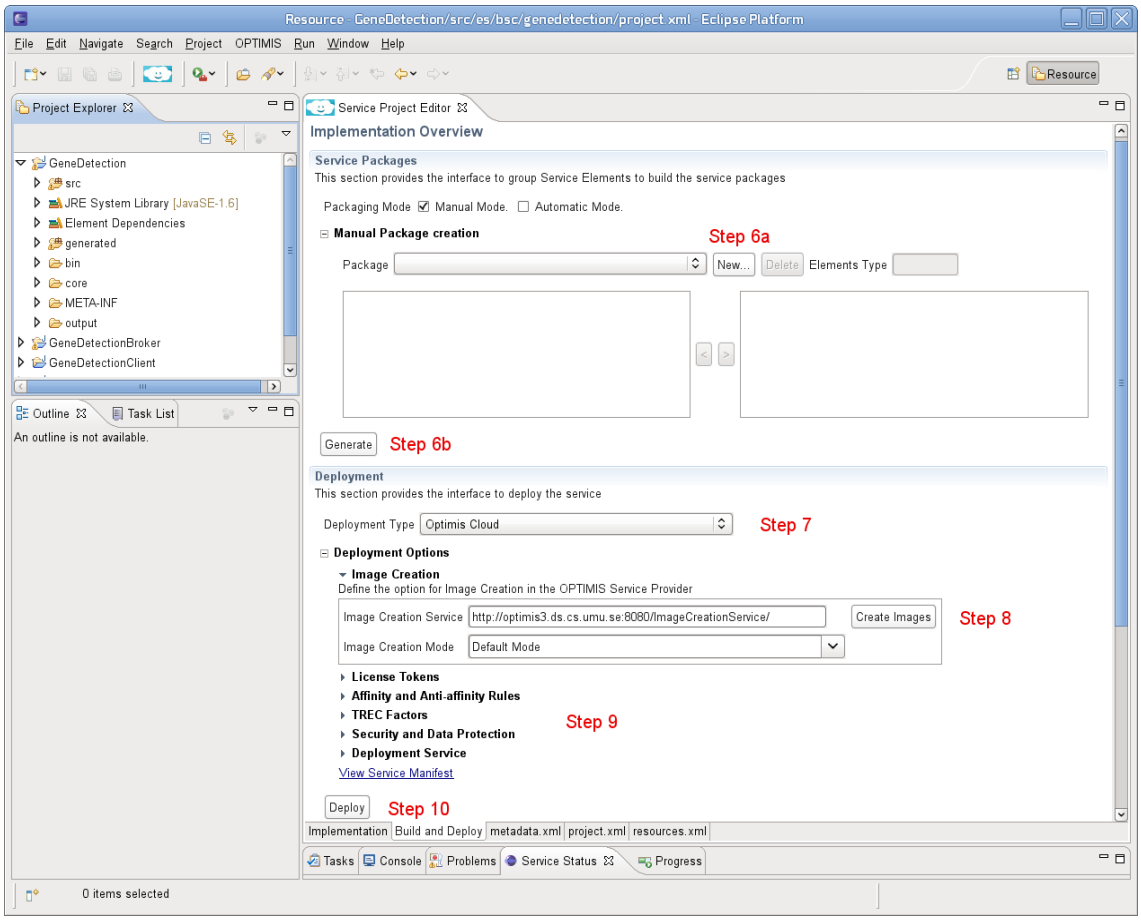

Figure 6. Building and Deployment Tab

Step 8.

Create Service VM images by opening the Image Creation options, setting the location of Image Creation Service and clicking the “Create Images” button. (Figure 6 Step 8)

Step 9.

Define TREC, Security & Data protection, Affinity & Anti-affinity rules between for the service packages. Generate license tokens in case of your service require them (Figure 6 Step 9)

Step 10.

Deploy in the OPTIMIS Cloud by clicking “Deploy” (Figure 6 Step 10)

2.5.2 Testing the Software

Tests are not currently provided

2.5.3 Configuration

Not required

2.6 Other information

2.6.1 Source Code Information

The source code of the IDE follows a structure of an Eclipse Plugin. The java classes which implement the plug-in are located in the src/main/java folder. These classes are organized in packages according to the part of the IDE plug-in they are implemented (editors, wizards, actions, dialogs, etc.)

The properties and configuration files in the Root, META-INF and OSGI-INF for defining and configuring the plugin and specify the different options of the building process.

The icons folder contains images and icons used in the IDE plug-in.

The lib folder stores the dependencies used by the plug-in. Some of these dependencies are automatically copied to this folder by the maven tool.

2.6.2 Contributors

Jorge Ejarque (Barcelona Supercomputing Center)

3 References

[1] Eclipse Indigo http://www.eclipse.org/indigo/
[2] OPTIMIS Detailed Design, Deliverable D1.2.2.2 of OPTIMIS project.
[3] Apache Tomcat http://tomcat.apache.org/