

# Integrated Development Environment

User Guide Version 1.2



# **Table of Contents**

TAB	TABLE OF CONTENTS				
1.	INSTALLING THE SOFTWARE	4			
1.1.	Installation from COMPSs Eclipse Update Site				
1.2.	Installation from Eclipse Marketplace	5			
1.3.	Installation from COMPSs VM	6			
1.4.	Installation from Sources	7			
2.	GETTING STARTED	Q			
2.	Step 1: Create a new COMPSs application project				
	Step 2: Create a new Orchestration Class.				
	Step 3: Add an Orchestration Element				
	Step 4: Add Core Elements				
	Step 5: Implement the Orchestration Element workflow				
	Step 6: Check the application locally.				
	Step 7: Deploy the application in a distributed infrastructure.				
3.	OTHER FUNCTIONALITIES	15			
3.1.	Implementation				
3.	1.1. Create a Core Elements				
	New Method Core Element from Existing Class				
	New Method Core Element from Executable				
	New Service Core Element from WSDL				
	New Service Core Element from WAR package				
3.	1.2. Invocation of Core Elements from the Orchestration Element Code.				
	Invoke Method Core Elements				
_	Invoke Service Core Elements				
	1.3. Manage Application Elements Constraints				
	1.4. Add Elasticity Boundaries to Application Elements				
	1.5. Manage Application Element Dependencies				
3.	1.6. Adding Multiple Implementations for a Core Element Add Implementation Specific Constraints				
	Add Implementation Specific Constraints				
	Invocation of Core Elements with Multiple Implementations				
3	1.7. Define a COMPSs Application from existing war and jar packages				
-	1.8.         Convert a Java Project to COMPSs Application Project				
3.2.	Deployment				
3.	2.1. Deployment Options				
	Deployment in Private Grid				
	Resource Selection				
	Define Deployment Information				
	Define Shared Disks				
	Define Data Staging				
	Deployment with the Enactment Service				
	Define Storage Service				
2	Define Enactment Service				
	2.2. Select the Main class				
KNC	OWN LIMITATIONS				
REFE	ERENCES	37			

# 1. Installing the software

Next paragraphs describe the different option for installing the COMPSs IDE plugin in the Eclipse platform [1].

# **1.1.Installation from COMPSs Eclipse Update Site**

To install the COMPSs IDE from the COMPSs Update Site, open the "*Install New Software...*" option of the Eclipse *Help* menu as shown in Figure 1.

۲		Reso
<u>F</u> ile <u>E</u> dit <u>R</u> un <u>N</u> avigate Se <u>a</u> rch <u>P</u> roject <u>W</u> indov	v <u>H</u> elp	
📑 🔛 📾 💼 💁 🕫 🔗	Welcome	
	Help Contents	
Project Explorer 🛿 🦳 🗖	S <u>e</u> arch	
⊟ 🔄 💝 🏹	Dynamic Help	
▶ 👺 AnalysisInvoker	Key Assist Shift+Ctrl+L	
▶ 🛃 ApplicationGlobalCheckings	Tips and Tricks	
▶	Report Bug or Enhancement	
	<u>C</u> heat Sheets	
🗄 Outline 🛿 🔋 Task List 👘 🌄 🗖	ZK Helps >	
An outline is not available.	Activate ZK Studio	
	Software Updates	
	Check for <u>U</u> pdates	
	Install New Software	
	Eclipse <u>M</u> arketplace	
	<u>A</u> bout Eclipse	

Figure 1. Snapshot for opening the wizard to install new software

Then, a new window like in Figure 2-left will be open. Click *"Add..."* to include the Eclipse Update Site for the COMPSs IDE. Afterwards, introduce the URL of our update site <u>http://compss.bsc.es/releases/ide/</u> in the new dialog window (Figure 2-right).

•		$\times$			
Available Software					
Select a site or enter the location of a site.					
0					
Work with: type or select a site	\ <b>\</b>	Add			
	Find more software by working with the <u>"Available Software Sites</u>	preferences.	۲	Add Repository	$\mathbf{X}$
type filter text		4			
Name	Version		<u>N</u> ame:	compss-ide	L <u>o</u> cal
There is no site selected.			Location:	http://compss.bsc.es/releases/ide/	<u>A</u> rchive
			?	Canc	el OK

Figure 2. Snapshots for adding the COMPSs IDE Update Site

Once the repository has been included you will see that the window has been updated with the available COMPSs IDE plugins (Figure 3). Select the *Core* plug-in and the desired *Deployers* plug-ins according to the distributed infrastructure you have available. Once the plugins are selected, click *"Next >"*, accept the license terms and click *"Finish"* to start the installation process. Once the installation process is complete the COMPSs IDE will be available on your Eclipse installation.

0	Install
Available So	oftware The second s
Check the it	ems that you wish to install.
<u>W</u> ork with:	compss-ide - http://compss.bsc.es/releases/ide/
	Find more software by working with the "Available Software Sites" preferences.
type filter tex	t 🖌
Name	Version
🗢 🗹 💷 CO	MPSs-IDE-Core
🗹 🖗	COMPSs-IDE-Core 1.1.0.201402141054
🕨 🖃 💷 CO	MPSs-IDE-Deployers
Select All	Deselect All 3 items selected
Details	
Show only	y the latest versions of available software $\Box$ Hide items that are already installed
☑ <u>G</u> roup iter	ms by category What is <u>already installed</u> ?
Show only	y software applicable to target environment
☑ <u>C</u> ontact a	II update sites during install to find required software
?	< Back Next > Cancel Einish

Figure 3. Available COMPSs IDE plug-ins

#### **1.2. Installation from Eclipse Marketplace**

To install the COMPSs IDE from the Eclipse Marketplace [2], open the "*Eclipse Marketplace...*" option of the Eclipse *Help* menu as shown in Figure 4.

•		
<u>F</u> ile <u>E</u> dit <u>R</u> un <u>N</u> avigate Se <u>a</u> rch <u>P</u> roject <u>W</u> indow	Help	
📬 🔛 🕼 🗁 🖬 💁 🖉	Welcome	
Project Explorer 🕱 🖓 🖓	Help Contents	
	S <u>e</u> arch <u>D</u> ynamic Help	
AnalysisInvoker	Key Assist Shift+Ctrl+L Tips and Tricks	
Bud SearchWeb	Tips and Tricks Report Bug or Enhancement	
CloudSearchWeb	<u>Cheat Sheets</u>	
🗄 Outline 🛛 🔋 Task List 🛛 🔋 🍷 🗖 🗖	ZK Helps >	
An outline is not available.	Activate ZK Studio	
	Software Updates	
	Check for <u>U</u> pdates	
	Install New <u>S</u> oftware	
	Eclipse <u>M</u> arketplace	
	About Eclipse	

Figure 4. Open Eclipse Marketplace snapshot

Then, a new window like in Figure 5 will be open where users can search for Eclipse extensions. Introduce "*COMPSs IDE*" in the text box and click "*Go*". You will see that a COMPSs entry appear in list of the view. Finally, click "*Install*" inside COMPSs IDE entry to start the installation process.

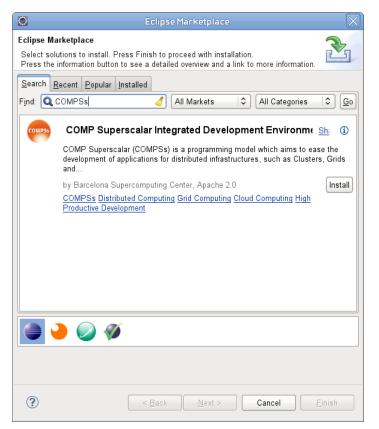


Figure 5. Eclipse Marketplace snapshot

From this point on, the installation is the same as the explained in Section 1.1. Users must select the IDE core feature and the desired *Deployers* plug-ins and follow the instructions provided by the installation wizard to accept the license terms and finalize de installation process.

# 1.3. Installation from COMPSs VM

The following instructions have been detailed for the Oracle Virtual Box Manager [3] version 4 or higher. To install the COMPSs Virtual Appliance you need to perform the following steps:

- 1. Download the Virtual Appliance file from the COMPSs web site [4].
- 2. Open the Virtual Box VM Administrator.
- 3. Select the *Import Virtualised service* option in the *File* menu and an import wizard will be open.
- 4. Click on *Select...* and choose the downloaded *.ova* file and click *Next*.
- 5. The following page will provide a default VM configuration for this appliance. In case some parameters are not suitable for your system change them.
- 6. When finish, click *Import* to start the VM import process.
- 7. After a successful import process, a new VM will appear in the main window of the VM Virtual Box Administrator.
- 8. Select the newly created VM and click *Start*.

The COMPSs IDE is installed as an Eclipse plug-in. The plugin has also prepared to work with a COMPSs runtime libraries installed in */opt/COMPSs/*. You have to set this location when a new COMPSs project is created with the IDE.

# **1.4.Installation from Sources**

For installing the COMPSs IDE plug-in from the source code you require to install first the following software:

- Eclipse platform (version 3.6 or higher) and the Web Development Tools plug-in
- The Subversion client [5] to check-out the source code
- Apache Maven [6] (version 3 or higher) to compile the source code

Once the software is installed in your machine, follow the steps described below to install the COMPSs IDE plug-in.

1.- Check-out the source code from the URL obtained from the COMPSs website[4].

user@host:~\$ svn checkout <u>http://<compss/svn/ide\_path</u>> <ide\_co\_path>

2.- Compile the code and build the plug-in using Maven.

user@host:~\$ cd <ide\_co\_path>
user@host:<ide\_co\_path>\$ mvn clean install -Dmaven.test.skip=true

3.- Initiate the eclipse platform.

user@host:~\$ ECLIPSE\_HOME/eclipse

4.- Install the plug-in from a local update site. Follow the same steps as explained in Section 1.1, but substituting the COMPSs update site URL by the following local URL:

jar:file<ide co path>/IDE-site/target/site-<version>-SNAPSHOT.zip!/

# 2. Getting Started

A Service developed by the COMPSs Programming Model is composed by orchestration and core Elements (a.k.a. Tasks). 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.

If the IDE plug-in has been successfully installed a *CompSs* menu with different actions should appear in the Menu bar and a set of *COMPSs wizards* should appear in the *"File->New..."* sub menu when the Eclipse platform is initiated. There is also an *Application Editor* which will be available when a new project is created. You can use these actions, wizards and editor to easily create a new COMPSs application with different orchestration and core elements in seven steps.

#### Step 1: Create a new COMPSs application project

The first step to implement an application with the COMPSs programming model is creating a new COMPSs application project. This can be done by opening the menu *"File ->New-> Project"* and selecting the *New Application Project wizard* located in the *COMP Superscalar section* as shows in Figure 6.

New Project	$\sim$
Select a wizard	
<u>W</u> izards:	
type filter text	4
<ul> <li></li></ul>	
Cancel	Einish

Figure 6. Creating a new COMPSs application project

Once the wizard is open, a window like in Figure 7 will appear. Introduce the desired name for the project, the name of the main package where the application classes will belong to and the location where the COMPSs libraries are installed in your machine. Finally, click *"Finish"* to start the process for creating the project.

New COMP5s Application Project
COMPSs Application Project
This wizard allows users to create a new COMPSs Application project
Project Name NewProject
Main Package project.main_package
Runtime Location
Runtime Location /opt/COMPSs Browse
< Back         Next >         Cancel         Finish

Figure 7. New COMPSs Application project wizard.

As result of this wizard, the IDE creates a new project and opens the *Application Editor* will as shown in Figure 8. In this figure, the actions required for some of the next steps has been also depicted.

6	Resource - NewProject/META-INF/metadata xml - Eclipse Platform	
<u>F</u> ile <u>E</u> dit <u>N</u> avigate Se <u>a</u> rch <u>P</u> roject Co	mpSs <u>R</u> un <u>W</u> indow <u>H</u> elp	
] 🗗 🖩 🖻 🛎 ] 🥽 🛛 🏎 ] 🛷	• 」 空く むく ひく ひく	😭 🔁 Resource
Project Explorer 🛿 🗖 🗖	en Application Editor 🛙	- 8
□ 😫 🎽	Implementation Overview	
NewProject	Orchestration Classes Runtime This section provides the interface to manage Orchestration Classes Step 2 This section provides the interface to manage the Runtime location.	
	Orch. Class V New Import Delete Location	Change
	▼ Orchestration Elements	
	This section provides the interface to manage the orchestration elements of the selected class. Step 3 This section provides the interface to manage the core elements of the selected class.	Step 4
	blow)	New
	Orchestration Element Description     Core Element Description	
E Outline 23 - D An outline is not available.		
	Step 6 and 7	•
	Implementation Build and Deploy  metadata xml  project.xml	
	a Tasks 🕴	~
	0 items v I Description Resource Path Location Type	
_ □•		

Figure 8. COMPSS IDE Service Editor

#### Step 2: Create a new Orchestration Class.

Once the project is created, developers have to include a new class to include Orchestration and Core Elements. This phase can be done with the *New COMPSs Application Class* wizard (such as Figure 9). This wizard will be opened by clicking the "*New…*" button located on the *Orchestration Classes* section of the *Implementation Tab* in the *Application Editor* (Figure 8. Step 2). Developers have to fill the *Name* field, select the class *Type*. The current available class types are standard Class or a Web Service Interface Class options and click "Finish". Then a new Orchestration Class and Core Element interface will be created in the main package of the project.

E	New COMP5s Application Class	X					
COMPSs Application Class							
Creates a new COMPSs Application class for including Orchestration Elements							
Source fol <u>d</u> er:	NewProject/src Browse	٦					
Pac <u>k</u> age:	project.main_package Browse	j					
Na <u>m</u> e:	NewWSClass						
Туре	Web Service Interface Class						
	Standard Class						
	Web Service Interface Class						
?	Cancel <u>E</u> inish	]					

Figure 9. New Orchestration Class wizard snapshot.

#### Step 3: Add an Orchestration Element

After the creation of the Orchestration Class, developers have to include an Orchestration Element in this class. This Orchestration Element will implement the workflow with different Core Element invocations. An Orchestration Element can be added by using the *New Orchestration Element* wizard (Figure 10) open when clicking the *"New..."* button of the *Orchestration Elements* section located on the *Implementation Tab* of the *Application Editor* (Figure 8. Step 3)

Developers have to define at least the name of the Orchestration Element method, the return type and the method parameters. If the Orchestration class is a WS interface we can also indicate if the new Orchestration Element is going to be part of the WS interface or a private method. It is done by checking the option *Part of the Service Interface*. If the Orchestration Element has special constraints, it can be also defined in the wizard like described in Section 3.1.3. Once the wizard is finalized, a new method is created in the orchestration class with the corresponding parameters and annotations.

•	New Orchestration Element	X				
New Orchestration Element						
Creates a new	Orchestration Element for an application					
Orchestration E	Element Location	]				
Source fol <u>d</u> er:	NewProject/src	Br <u>o</u> wse				
Pac <u>k</u> age:	project.main_package	Bro <u>w</u> se				
Orch. Class	NewWSClass	Bro <u>w</u> se				
Orchestration E	Element Description					
Name	newOEMethod	Select				
Return Type	void	Select				
Parameters	Type Name java.lang.String input	Add Modify Delete				
Constraints	Name Value	Add Modify Delete				
☑ Part of the	Service Interface					
?	Cancel	Einish				

Figure 10. Orchestration Element Wizard

#### Step 4: Add Core Elements

The core elements invoked from the orchestration elements of an orchestration class must be defined in the corresponding Core Element Interface. To do it, the COMPSs IDE offers a *New Core Element* wizard, which is open by clicking the *"New..."* button of the *Core Elements* section located on the *Implementation Tab* of the *Application Editor* (Figure 8. Step 4). Developers currently have several options to create a new core element as depicted in Figure 11. In this part of the document, we are just going to explain detail how to create a new method core element from scratch. More details about the other options for creating core elements are detailed in Section 3.1.1.

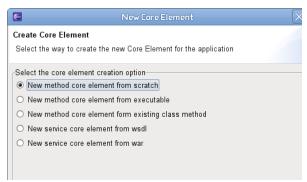


Figure 11. New Core Element wizard snapshot

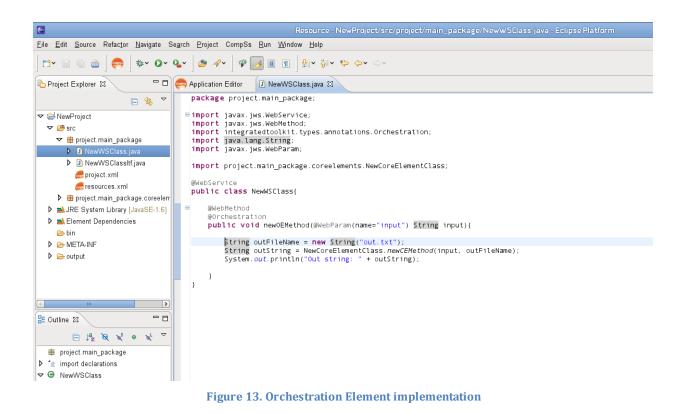
When a developer selects the option for creating a new core element from scratch, the wizard will show a view (Figure 12-left) to introduce the class where the new core element will be included and the desired method name for the new core element. Then, after clicking *"Next"*, the wizard will show a view (Figure 12-right) to complete the required element information (return type, parameters and constraints). These values are similar than in the ones specified for *New Orchestration Element* wizard with the different that core elements parameters must define the direction of the parameter (*IN*, *OUT* or *INOUT*) and indicate if the parameter is a file with setting the parameter type as *Type.FILE*. Finally, the core element creation is finalized by clicking *"Finish"*, and the new core element method is created in the Core Clement Interface and the declaring class.

E	New Core Element	$\sim$	6		New Core E	lement	$\times$
Create Core El Define the Core	ement Element with the selected procedure		New Core Ele Creates a new		nt for an applicatio	n	
Pac <u>k</u> age:	ion NewProject/src project.main_package NewWSClassItf	Br <u>o</u> wse Bro <u>w</u> se Bro <u>w</u> se	Element Loca Source fol <u>d</u> er Pac <u>k</u> age: CE Interface	NewProjec	in_package		Browse Browse Browse
Element descr Declaring Cla Method	iption ss NewCoreElementClass newCEMethod		Element desc Name Return Type	newCEMeth			Select
			Parameters	Type Type.FILE Type.FILE	Name inputFile outputFile	Direction IN OUT	Add Modify Delete
			Constraints	Name Valu	90 J		Add Modify Delete
			✓ Specific C Declaring Cl Options Method Mod	ass NewCo	oreElementClass nit ☑ isModifier		
?	< <u>B</u> ack <u>N</u> ext > Cancel	Einish	(?)	< <u>F</u>	<u>J</u> ack	Cancel	Einish

Figure 12. New Core Element Wizard from scratch snapshots.

#### Step 5: Implement the Orchestration Element workflow

Once the Core Elements have been defined, the developer has to program the business process or workflow that implements the application functionality inside the Orchestration Element method. An example code is depicted in Figure 13. This workflow is programmed as a sequential code where the different defined Core Elements are invoked as you will do with in standard Java. At runtime, COMPSs will detect the data dependencies between the invocations and execute tasks concurrently according to those data dependencies. More details can be found at Section 3.1.2.



## Step 6: Check the application locally.

To check an application developed with the COMPSs runtime keeps the correct behaviour, developers can perform a local deployment and execution of the application. It is performed by selecting the *Build and Deploy Tab* in the *Application Editor* (Figure 8. Steps 6 and 7) and then, selecting the *Localhost* deployment type.

Figure 14 shows a snapshot of the *Application Editor* for a local deployment. To deploy the application locally, developers only have to define the folder to install the application elements (*App. Elements folder* field) and if the application contains WS classes the location where a Web Application Server (such as Apache Tomcat) is installed on the local machine (*App. Server Folder* field). Finally, developers have to click deploy to start the deployment process.

Once the deployment process is completed, developers will see a Deployment view with the machines where the elements have been deployed (*localhost* in this case), and can also find the generated packages in the output folder of the project and their installation in the selected folder.

G					
Eile Edit Navigate Search Project CompSs Run Window Help					
] 🗗 🖬 🖻 📥 ] 🤤 ] 💁 ] 🛷	<ul> <li>         ・</li></ul>	🗈 🕞 Resource			
ြာ Project Explorer 🛿 🦳 🗖	i Application Editor 🕴 👔 NewWSClass java	- 0			
😑 🐐 🎽	Deployment Overview				
	Deployment This section provides the interface to deploy the application in a distributed computing platform Deployment Type Deployment Options Ann Saver Folder Minmadignee/TestReds/INET_Testhed/secret/ Select the leading to deploy the				
	Tasks      Console      Application Status 83	- 0			
	ai	deploy the pplication			
□ ♥ 0 items selected					

Figure 14. Local deployment snapshot

#### Step 7: Deploy the application in a distributed infrastructure.

Once the developer has checked the correct application behaviour including COMPSs, it is time to make the final deployment of the application in a distributed infrastructure. For doing it, you can select one of the other deployment options and follow the specific instructions for the selected deployment environment explained in detail in Section 3.2.1.

# 3. Other Functionalities

In addition to the core functionalities described in the *Getting Started* section, the COMPSs IDE provides other functionalities to simplify the implementation of applications with the COMPSs programming model. We have grouped these extra functionalities in two parts: the first part covers the implementation phase and the second part which covers the available deployment options.

# 3.1. Implementation

# 3.1.1. Create a Core Elements

Once an orchestration class is created a user can create a new core element, selecting the orchestration class in the combo box and clicking *"New..."* in the *Core Elements* section of the *Application 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 type and constraints, and the wizard creates the class methods and required annotations. This is the case explained in the *Step 4* of the *Getting Started* section.
- **Method Core Element from an existing class**, where the user selects a method from an existing class, adding only the constraints, and the wizard generates the core element definition in the Core Element Interface
- **Method Core Element from Executable,** where the users select the command or binary and their arguments, and the wizard generates the code to invoke the executable and add the core element definition in the Core Element Interface.
- **Service Core Element from WSDL**, where the users select a method from a deployed web service, and the wizard generates the service stubs and includes the service core element definition in the Core Element Interface.
- **Service Core Element from a war package,** where the users select package location and the web service method included in this package, and the wizard generates the service stubs and includes the service core element definition in the Core Element Interface.

# New Method Core Element from Existing Class

When a developer selects the option for creating a new core element from an existing class, the wizard will show a view (Figure 15-left) to introduce the library or class folder location, select one of the existing classes and select the method for the new core element. Then, after clicking *"Next"*, the wizard will show a view like Figure 15–right. There, some default data will be already filled and developers only have to modify the desired parameters type and direction (if they are different from the default values) and include constraints if it is required.

E	New Core Element	$\sim$	E	N	lew Core	Element		$\times$	
Create Core Ele Define the Core	ement Element with the selected procedure	New Core Element Creates a new Core Element for an application							
Element Locati	on		Element Loca	tion					
Source fol <u>d</u> er:	NewProject/src	Browse	Source fol <u>d</u> er:	: NewProject/src			Br <u>o</u> wse		
Pac <u>k</u> age:	project.main_package	Browse	Pac <u>k</u> age:	project.main_pac	ckage			Bro <u>w</u> se	
CE Interface	NewWSClassItf	Browse	CE Interface	NewWSClassItf				Bro <u>w</u> se	
Element descri	ption		Element desc	ription					
Library Locati	on /home/jorgee/Apps/Matmul/MatrixMultiply.jar	Select	Name	multiplyAccumulat	ive				
Declaring Cla	ss worker. matmul. Matmulimpi	Select	Return Type	void				Select	
				Туре	Name	Direction		Add	
Method	multiplyAccumulative	Select	Parameters	java.lang.String	fЗ	IN	=	Modify	
				java.lang.String java.lang.String	f1 f2	IN IN	~	Delete	
				Java.lang.othing	112	1.04			
				Name Value			Add		
			Constraints					Modify	
								Delete	
			✓ Specific C	ore Element Desci	ription				
			Declaring Cla	worker.matm	nul.Matmul	Impl			
			Options	🗆 islnit 🗹	isModifier				
		Method Mod	ifiers 🗹 static 🗌	final					
?	<pre>&lt; Back Next &gt; Cancel</pre>	<u>E</u> inish	?	< <u>B</u> ack	<u>N</u> e:	xt > Ca	ncel	<u>F</u> inish	

Figure 15. New Core Element from existing class snapshots

# New Method Core Element from Executable

When a developer selects the option for creating a new Core Element from Executable (binary command, script...), the wizard will show a view (Figure 16-left) to introduce the information to bind the Method Core Element interface with the command to run the executable. First, the developer has to introduce the name of the class and method where, the IDE is going to create the code to run the executable. Once it has been defined, developers have some fields to specify the executable the arguments and the standard streams (*stdin, stdout, stderr*). In this fields, we can add a reference to parameters or return values of the CE method by introducing the pattern *\$<parameter\_name>\$* or *\$return\_type\$* respectively.

For instance, the second argument of the CE defined in Figure 16-left will contain the value of the CE method parameter called *arg\_from\_params*. This figure also shows how to return the standard output of the executable as a *return value* of the CE method.

Then, after clicking *"Next"*, the wizard will show a view (Figure 16 - right). This is the classical view for describing a core element. Note that the IDE has already detect the method should include a parameter called *arg\_from\_params* and a *String* return type for passing the standard output. As in the other cases, developers can complete the required element information such as changing the parameters types and directions as well as defining the constraints required by the executable command.

E	New Core Element	New Core Element
Create Core E Define the Core	lement a Element with the selected procedure	New Core Element Creates a new Core Element for an application
Pac <u>k</u> age: CE Interface Element desc	NewProject/src     Browse       project.main_package     Browse       NewWSClassttf     Browse	Element Location         Source folder:       NewProject/src         Package:       project.main_package         Browse         CE Interface       NewWSClassItf         Browse         Element description         Name       runBinary         Return Type       java.lang.String         Select         Type       Name         Add
← Executabl		Parameters     java.lang.String     arg_from_param     IN     Modify       Delete     Image: String     Add       Constraints     Image: String     Add
	Streams	Delete
Standard In Standard Or Standard Er	utput \$return_type\$	▼ Specific Core Element Description         Declaring Class       project.main_package.coreelements.BinaryCE         Options       □ isInit ☑ isModifier         Method Modifiers       ☑ static □ final
?	< Back Next > Cancel Einish	? < Back Next > Cancel Einish

Figure 16. New Core Element from Executable snapshots.

After making the required changes, developers have to click "Finish" to perform the core element creation. In this case, in addition to include the method declaration in the Core Element Interface, the IDE also generates the code for invoking the executable file. Figure 17 shows the code generated for the previous example where you can see how the IDE also link the CE parameters with the arguments and standard output of the command to run.

16	e Application Editor	D BinaryCE.java 🛛	- 0
	package projec	t.main_package.coreelements;	<u>^</u>
	public class B	inaryCE{	
	java.l: String[ cmd[0] cmd[1]	<pre>tic String runBinary(String arg_from_param){ ang.String return_type = null;</pre>	-
	Proces: try { in	<pre>s execProc = null; sBuilder pb = new ProcessBuilder(cmd); t exitValue = 0; r (int i = 0; i &lt; 10; i++) { System.out.println("Attempt " + i + " out of " + 3); execProc = pb.start(); execProc.getOutputStream().close();</pre>	=
-		<pre>java.io.BufferedInputStream stderr_is_bis = new java.io.BufferedInputStream(execPro byte[]stderr_is_bis.read(stderr_is_b) &gt;= 0); stderr_is_bis.close(); execProc.getErrorStream().close(); java.io.InputStream stdout_is = execProc.getInputStream(); StringBuilder return_type_sb = new StringBuilder(); java.io.BufferedReader stdout_is_br = new java.io.BufferedReader(new java.io.Input String stdout_is_line; while ((stdout_is_line = stdout_is_br.readLine()) != null) return_type_sb.append(stdout_is_line); stdout is br.close();</pre>	
	<		>

Figure 17. Snapshot of generated for creating a CE from an executable

#### New Service Core Element from WSDL

When a developer selects the option for creating a new Core Element from a WSDL, the wizard will show a view like in Figure 18–left. This view provides the interface to introduce the location where the service WSDL is deployed. Once the WSDL is loaded, we can select one of the services, ports and methods contained in this WSDL file. Then, after clicking *"Next"*, the wizard will show a view (Figure 18-right) to complete the required element information like in other core elements.

	New Core Element	$\left  \right\rangle$	6	New Core Element	$\times$			
Create Core Element         New Core Element           Define the Core Element with the selected procedure         Creates a new Core Element for an application								
	iewProject/src	Browse Browse Browse vsdl Select Select	Pac <u>k</u> age: CE Interface Element descri Name Return Type	NewProject/src project.main_package NewWSClassItf	Browse Browse Browse Select Add Delete			
Method	(parseBlastIDs	Select	Constraints	Name Value Dre Element Description BioTools (http://ganedetect.core BioToolsPort	Add Modify Delete			
?	< Back Next > Cancel	Einish	Locations	http://bscgrid05.bsc.es.20390/biotools/biotools?ws < Back Next > Cancel	Add Modify			

Figure 18. New Core Element from WSDL snapshots.

After making the required changes, developers have to click *"Finish"* to perform the core element creation. In this case, in addition to include the method declaration in the Core Element Interface, the IDE generates the classes used by the service as well as the stubs to invoke the service from the Orchestration Element. The generated classes can be found in the generated folder of the COMPSs Application project as depicted in Figure 19.

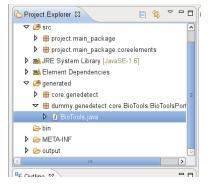


Figure 19. Generated class snapshot.

#### New Service Core Element from WAR package

The option for creating a new Core Element from a WAR package is very similar than the previous option. When a developer select this option, the wizard will show a view like Figure 20–up-left. There, the developer has to select the location where the WAR package is in the local host. Then, the IDE will extract the package, and afterwards the developer has to select the WSDL file (Figure 20-down-left) and URL pattern which contains the service description. Finally, the developer has to select the Service, port and method like in the previous case (new CE from WSDL). Then, after clicking *"Next"*, the wizard will show a view (Figure 20-right) to complete the required element information (return type, parameters and constraints).

E New Core Element	K New Core Element	$\mathbf{X}$		
Create Core Element	New Core Element			
Define the Core Element with the selected procedure	Creates a new Core Element for an application			
Element Location	Element Location			
Source folder: NewProject/src Browse	Source folder: NewProject/src Br	r <u>o</u> wse		
Package: project.main_package Browse	Package: project.main_package Br	ro <u>w</u> se		
CE Interface NewWSClassItf Browse	CE Interface NewWSClassItf Br	ro <u>w</u> se		
Element description	Element description			
WAR Location /home/jorgee/Apps/GeneDetection/biotools.war Select	Name loadAminoAcidSequence			
WSDL Location //home/jorgee/clean_workspace/NewProject/output) Select	Return Type void	Select		
URL pattern biotools/* Select	Type Name Direction	Add		
	Parameters core.genedetect.NemusObject seqId IN	Modify		
Service BioTools Select		Delete		
Port BioToolsPort Select				
Method IoadAminoAcidSequence Select		Add		
		Modify		
Select WSDL File		Delete		
	Specific Core Element Description			
NewProject output war_tmp WEB-INF wsdl	Service Name BioTools			
Lugar: BioTools.wsdl	Service Name http://genedetect.core			
Lugares Nombre V Tamaño Modificado	Port Name BioToolsPort			
Q Buscar E BioTools.wsdl 5,3 KiB 08/05/13	Locations	Add		
i jorgee ∎		1odify		
Sistem				
Añadir Quitar				
<u>C</u> ancelar <u>A</u> ceptar	Cancel	<u>F</u> inish		

Figure 20. new Core Element from WAR Packages Snapshots.

#### 3.1.2. Invocation of Core Elements from the Orchestration Element Code.

Once the developer has defined the Core Elements in the Core Element Interface, the next step is invoking them form the Orchestration Element code. As a general rule, the invocation of core elements is done in the same way as you call a normal Java method. In next paragraph, we are going to refresh how to invoke them, including some details you must take into account when programming the Orchestration Elements.

#### Invoke Method Core Elements

In Java there are main types of methods: Class and Object methods. Class methods are static methods which make their computation only using the method arguments or static class properties. They are invoked as the example shown below:

```
ClassName.staticMethodName(argument1, argument2);
```

Object methods use the values of an object properties in addition to the method arguments to perform the computation and can also modify these object properties. To invoke this method you need to have an object to perform the method call. An example of this type of calls is shown below:

ClassName object = new ClassName();

Object.objMethodName(argument1, argument2);

For this type of call, the COMPSs runtime introduce an implicit parameter, which is the object where the call is performed. By default, this parameter is treated with an INOUT direction because every object method can update whatever object property. This implies that the object could be different at the end of the method execution and the runtime must be aware of this. If developers know that a method, defined as core element, does not modify the object properties, we recommend marking this core element as *Not Modifier* in the Core Element Interface. It can be done by including the *Method* annotation property *isModifier* equals to *false*. If it modifies the object or the developer is not sure, keep the default value (*isModifier* = *true*) to ensure the application is working properly.

#### Invoke Service Core Elements

When we define a service core element, the IDE creates the classes to invoke the service methods from the orchestration elements code. Most of the created classes are the data types used as arguments of the method services, but we also generate a "*dummy*" stub to make the method invocation. We call it "*dummy*" because the code of this stub will never be executed. The runtime will intercept the call to this stub and will perform the real invocation of the remote service. As java methods, there are two types of web services calls: *stateless* and *stateful*.

On one hand, the *stateless* service calls invoke method whose execution does not modify the service state. So, we could invoke them in the orchestration code as static calls by invoking the method in the *Static* version of the service class. An example is shown below:

```
import org.namespace.ServiceName.ServicePort.*;
import org.namespace.DataTypeClass;
...
DataTypeClass result = ServiceName.Static.statelessMethodName(arg1, arg2);
```

On the other hand, the *stateful* service calls invoke methods that modify the service state. This must be reflected in the orchestration code in order to make a proper management of dependencies between the web service calls. Note that *stateful* service calls have a lot of similarities with the object method calls. For this reason, you have to use the same procedure

in both cases. To indicate a sequence of service calls are *stateful* you must invoke it as a

```
service object method. An example is shown below:
    import org.namespace.ServiceName.ServicePort.*;
    import org.namespace.DataTypeClass;
    ...
    DataTypeClass result = ServiceName.statefullMethodName(arg1, arg2);
```

#### 3.1.3. Manage Application Elements Constraints

Once an application element (Orchestration or Core) is created it can be selected in the corresponding *Orchestration/Core Elements* section of the *Application Editor*. The description of the element will be printed in the *Orchestration/Core Element Description* section. This section contains a *General Description* expandable item, which contains a *Constraints* table and a set of buttons to add, modify and delete constraints (Figure 21-left).

Application Editor					• #	
	ent Description	to manag	ge the selected core	alamant		
✓ General D		e to manag	ge the selected con	element.		
Name	multiplyAccumulat	tive				
Return Type	void			Select		
	Туре	Name	Direction			
	java.lang.String	f3	IN			
Parameters	java.lang.String	f1	IN	Modify		
	java.lang.String	f2	OUT		=	Name processorArchitecture 🗸
						Value x86_64
	Name	V	/alue	Add		
Constraints	processorArchited	cture x	(86_64	Modify		Cancel OK
				Delete		
► Specific D	escription					
▶ Elasticity E	Description					
▶ Dependen	cies Description					
					~	
lementation Build	10.1			>		

Figure 21. Add Constraint to Application Element Snapshots.

Developers can add constraints by clicking "Add..." and a new window (as depicted in Figure 21-right) will be open to select the type of constraint and set the value of the selected constraint. In a similar way, developers can modify the constraints values by selecting the desired constraint in the Constraints table and clicking "Modify..." and window to modify the constraint value will be open. Finally, to delete a constraint, developers only have to select the constraint in the Constraints table and click "Delete".

#### 3.1.4. Add Elasticity Boundaries to Application Elements

Once an application element (Orchestration or Core) is created, it can be selected in the corresponding *Orchestration/Core Elements* section of the *Application Editor*. The description of the element will be printed in the *Orchestration/Core Element Description* section. This section contains an *Elasticity Description* expandable item, where developers can specify the maximum and minimum number concurrent core element executions (Figure 22).

Figure 22. Elasticity Description Snapshot.

#### 3.1.5. Manage Application Element Dependencies

The application element implementations can depend to external libraries or require binaries or configuration files to run correctly. For this reason, developers can define dependencies in the application element descriptions. So, when an element is deployed in a distributed environment their dependencies will be also deployed to the selected location.

<ul> <li>Core Element Description</li> </ul>		
This section provides the interface to manage the selected core elemen	it.	
General Description		
Specific Description		
<ul> <li>Elasticity Description</li> <li>Dependencies Description</li> </ul>	7	Type JAR Library
Type Path JAR Library /home/jorgee/Apps/Matmul/MatrixMultiply.jar	Add	Location //MatrixMultiply.j; Select
	Delete	Cancel OK

Figure 23. Dependency Description Snapshots.

Once an application element (Orchestration or Core) is created, it can be selected in the corresponding *Orchestration/Core Elements* section of the *Application Editor*. The description of the element will be printed in the *Orchestration/Core Element Description* section. This section contains a *Dependency Description* expandable item (Figure 23), where developers can specify the packages, folders or files required to successfully execute an application element.

## 3.1.6. Adding Multiple Core Element Implementations

To define a new implementation of an existing Core Element, developers have to select the Core Element of the *Core Elements* section in the *Application Editor*. Then, the description of the selected Core Element will appear in the *Core Element Description* section. Open the *Specific Description* part and you will see an *Implementations* table like the depicted in Figure 24. Click *Add...* and a new window will appear like the one depicted in the figure. There, you can select the class that is implementing the new version of the Core Element and specific implementation constraints and dependencies. To finalize the Implementation definition, developers just need to click *OK*, and the new Core Element implementation will appear on the table.

	re Element E		selected core element							
This section provides the interface to manage the selected core element.      General Description										
	oecific Descr									
		Declaring Class	Constraints	Depend	lencies		Add			
Impl	lementations	es.bsc.hmmerpfam.coreeleme					Modify			
							Delete			
							Boloto			
	E				$\times$					
) E ) D	Declaring Cla	ass [			Select					
• •										
		Name Value			Add					
	Constraints				Modify					
					Delete					
		Type Path			1					
	Dependencie				Add					
	Dependencie				Delete					
				Cancel	ок					

Figure 24. Snapshot of Adding/Modifying Core Element Implementations

The changes made with the IDE will be reflected in the Core Element Interface of the application. Although the method signature remains, having implementations of the method in multiple classes produces changes on the *@Method* annotation of the CE. Instead of defining a single class on its *declaringClass* attribute, the annotation may define many of them; thus, instead of a single class the programmer defines a list of classes as the following example depicts:

## Add Implementation Specific Constraints

To define constraint to a specific Core Element implementation, the developer has to select the Core Element on the *Core Elements* section. When the description of the selected Core Element appears in the *Core Elements Description* section, open the *Specific Description* part, and the available implementations will be shown in the *Implementations* table as depicted in Figure 24. Select the Implementation and click *Modify...*. A new window will appear like the one depicted in the figure. To add a new constraint, developers just need to click on *Add...*, select one of the available constraints and set the desired constraint value like depicted in Figure 25. To finalize the constraint definition, developers just need to click *OK* in the constraints definition window and the new constraint will appear in the Constraints table. Once the implementation modifications have been finished, click *OK* in the Implementation definition window.

E		$\mathbf{X}$
Name	processorArchitecture	~
Value	x86_64	
	Cancel OK	

Figure 25: Constraint Definition Snapshot

#### Add Implementation Specific Dependencies

The procedure to define dependencies to a specific Core Element implementation is almost the same as above. The developer has to select the Core Element on the *Core Elements* section and open the *Specific Description* part of the *Core Elements Description* section. The available implementations will be shown in the *Implementations* table as depicted in Figure 24. Select the implementation and click *Modify...*. A new window will appear with a section for defining dependencies. To add a new dependency, developers just need to click on *Add...*. It will open a new window (Figure 26) where developers can specify the packages, folders or files required to successfully execute the core element implementation. To finalize the dependency definition, developers just need to click *OK* in the dependency definition window and the new dependency will appear in the *Dependencies* table. Once the implementation modifications have been finished, click *OK* in the *Implementation Definition* window.

Type JAR Library
Location [/MatrixMultiply.j;] Select]
Cancel OK

Figure 26. Snapshot of the widow for defining dependencies

## Invocation of Core Elements with Multiple Implementations

When developers define multiple implementations for the same CE, they are staging that all those methods are equal. In the orchestration element code, developers can invoke any of the methods defined as implementations. When the runtime receives one of the implementation calls, it creates an abstract task for the CE, and the implementation that is actually run is decided just before the execution according to the current application load and the infrastructure status.

#### 3.1.7. Define a COMPSs Application from existing war and jar packages

Once a COMPSs application project is created, a developer can import an application from existing packages by importing orchestration classes, defining orchestration and core elements from the classes and libraries contained in those packages.

To import an orchestration class, click *"Import..."* in the *Orchestration Classes* section of the *Application Editor*. Then, a new wizard, like Figure 27, will be open. There, developers can select the location of the package which contains the orchestration class. This is performed by clicking the "Select..." button in the *Package Location* field, and selecting the package file.

If the orchestration class to import is within a ZIP or WAR package, the IDE extracts the package in the *imported/<PackageName>* folder. Then, developers have to indicate the JAR library or class folder inside the extracted package. This is done by clicking the "Select..." button of the *Sub-package* field and selecting the location of the JAR file or the class folder.

In the last step, developers have to select the orchestration class by clicking the *"Select..."* button and selecting one of the classes in the list. After that, the selected class will appear in the *Orchestration Classes* section of the *Application Editor*.

│ <b>☆</b> ~ 🛛 🗟 │ 🏇~ (	6	Import COMPSs Application Class	×
Package Explorer 🛿		plication Class to import plication Class to import	
<ul> <li>Conversor</li> <li>✓ JavaProjectTest</li> <li>✓ ④ src</li> <li>✓ ⊕ java.project.test</li> <li>▷ ☑ MainClass.java</li> <li>☑ project.xml</li> <li>☑ resources.xml</li> </ul>	Package Location Sub-package (if any Orch. Class	(/home/jorgee/clean_workspace/GeneDeter jar) (/home/jorgee/clean_workspace/javaProjec com.jhlabs.image.AverageFilter	
	Co Mat © © ©	Class Selection	New
€ metadata.xml ⇒ output	Imi Dite	<sup>™</sup> ImageUtils m.jhlabs.image - /hoEB-INF/lib/Filters.jar Cancel OK	rml resources.xml

Figure 27. Import an Orchestration Class snapshot.

Once the orchestration class is imported, developers can define orchestration elements from the methods within the imported class. This process is started by clicking *"New"* in the *Orchestration Elements* section of the *Application Editor*. A new window, like Figure 28, will appear. There, developers can select the method to be defined as Orchestration Element, and specify the element constraints.

🛱 Package Explorer	×	🥮 Ap	E			X
	□ 🔄 🎽	Imple	Method			Select
👕 Conversor		Orch				F
🗢 🖼 javaProjectTest		This s		Name Value		Add
🗢 🗁 src		Orch				
🕨 🌐 com.jhlab	s.image	0.01	Constraints			Modify
🗢 🌐 java.proje	ct.test	- Or				Delete
🕨 🚺 MainC	lass.java	This s				Т
🛲 proje	ŧ	Motho	d Selection			c
🛲 reso		metho	abelection		Cancel	OK
🕀 java.pro	Choose a Method f	rom the s	elected class			
JRE Syste	b					
マ 🛋 Element D						
🕨 🔤 compss	S bias(float, float)					
🕨 🕮 importe	S bilinearInterpol	late(float,	float, int, int, i	int, int)		
🕨 🔤 Filters.j	S brightnessNTS	GC(int)				
🕨 🗁 bin						1
🕨 🗁 imported						
INF 🗁 META-INF						
🛲 metada						
🗁 output					i.xml project.xml resources.	xml
					ition	
			Cancel	ОК	Resource Path	L

Figure 28. Add Orchestration Element in imported Class snapshot

For the Core Elements, the process will be the same as the creating New Core Elements from existing classes. See more details of this process in Section 3.1.1.

## 3.1.8. Convert a Java Project to COMPSs Application Project

Another option to create applications with the COMPSs Programming Model and IDE is selecting or creating the orchestration and core from an existing Java project. To do this, developers have to convert the existing Java project into a COMPSs application project and afterwards, select or create the core elements as indicated in previous sections.

The IDE has a wizard to convert a Java project to COMPSs application project. This wizard can be started in two ways: one selecting the project in the Eclipse project explorer, click the left-button of the mouse, and select *COMPSs->Convert to Application Project* in the pop-up menu (Figure 29-left); and another opening the *CompSs* menu and selecting the *Implementation-> Convert to Application Project* (Figure 29 -right).

🖁 Package B	Explorer 🛛 🗖 🗖							
	- ⊈ ▼							
Conver								
🕨 😅 javaPre	viactTast							
	New	>						
	Go Into							
	Open in <u>N</u> ew Window							
	Ope <u>n</u> Type Hierarchy	F4						
	Sho <u>w</u> In	Shift+Alt+W >						
	<u>C</u> opy Copy Qualified Name	Ctrl+C			E			Java - Eclipse Pl
	Paste	Ctrl+V			<u>File Edit Source Refactor N</u> avigate S	e <u>a</u> rch <u>P</u> roject	CompSs <u>R</u> un <u>W</u> indow	Help
	Delete	Delete			<b>⊡</b> • 🗄 🗟 🚔 🍫 O• Q•		Implementation >	Create Orchestration Class
	Build Path	>			] 🗅 • 📾 🐨 🛯 🖗 • 💽 • 💁 ]	₩ G• ] 🖉	w.lki.e.e.	Add Orchestration Element
	Source	Shift+Alt+S >			🗏 Package Explorer 🛛 🗖 🗖			Add Core Element
	Refactor	Shift+Alt+T >						
	Import				□ 😫 🎽			Create Application Project
	Export				Conversor			
	Refresh	F5			🕨 😅 java Project Test			
	Close Project							
	Assign Working Sets							
	Run As	>						
	Debug As	Ś						
	Profile As	>						
	⊻alidate		@ Javadoc 😣 Declaration					
	T <u>e</u> am	>						
	Comp <u>a</u> re With	>		Resource				
	Restore from Local History.							
	COMPSs	$\rightarrow$	Convert to COMPSs Applic	ation Project				
	Checkstyle	>						
	Configure	>						
	P <u>r</u> operties	Alt+Enter						

Figure 29. Snapshots for starting the wizard for convert project to a COMPSs application project.

As result of these actions, the IDE will open the wizard (Figure 30), with the difference that, in the first case, the selected project will appear in Project Name, while in the second case, the user has to select the project from a list.

New COMPSs Application Project	$\mathbf{X}$
Convert Project from Java to COMPSs Application	
This wizard allows users to covert a Java project to a COMPSs Application project	
Project Name [javaProjectTest	Select
Main Package java.project.test	Select
Runtime Location	
Runtime Location /opt/COMPSs	Browse
Cancel	<u> </u>

Figure 30. Convert to COMPSs Application wizard snapshot.

Once developers have converted the project, they can also convert one of the current project classes into an orchestration class following the same procedure as explained in Section 3.1.7 for importing an orchestration class. To do it, developers have to click *"Import..."* in the *Orchestration Classes* section of the *Application Editor* and the import wizard will be open. If we do not define any package and click directly to the *"Select..."* button of the *Orch. Class* field, the *Class Selection* dialog will show the existing class in the converted project (Figure 31). So, developers can select one of them to convert it into an orchestration class.

	参~ (	Import COMPSs Application Class	Ì
🛱 Package Explorer 3	8	Select COMPSs Application Class to import	
	E	Select COMPSs Application Class to import	
<ul> <li>Conversor</li> <li>         ✓ javaProjectTest         <ul> <li>✓ # src</li> <li>♦ # com.jhlab:</li> <li>✓ # java.project</li> <li>♦ J MainCl</li> </ul> </li> </ul>	ct.test	Package Location     Select       Sub-package (if any jar)     Select       Orch. Class     Select	Dele
<pre></pre>	Choose	Class Selection       a class from the list       gitems:       Cancel       Einish	New
ि metadat ≥ output	🖶 java	t Description Peploy metadata.xml project.xml resources.x idoc @ Declaration a.project.test - javaProjectTest/src	ml
< <u> </u>	?	Cancel OK "	

Figure 31. Snapshot for defining an existing class as orchestration class.

Once the existing class has been converted, developers can create new orchestration element (as explained in Section 2. Step 3) or can select an existing method to be defined as orchestration element. To perform the second option, developers just have to click the *"New..."* button of the *Orchestration Elements* section located on the *Application Editor* and click "Select..." in the *Name* field of the *New Orchestration Element* wizard (Figure 32). Then, a *Method Selection* dialog will be open, and developers can select one of the methods declared in the converted class.

At this point, the converted project, class and orchestration element can be considered as if they were created from scratch. So, definition of core elements and other functionalities can be applied as described in the different sections of this guide.

G		Designation /META INE /			2)	
<u>F</u> ile <u>E</u> dit <u>N</u> avigate Se <u>a</u> rch [	🥃 New Orchestration Element 🛛 🗙					
_	New Orchestra Creates a new	<b>tion Element</b> Orchestration Element for an ap	plication			🗈 🖏 Java 🖒 Res »
<ul> <li>Conversor</li> <li>JavaProjectTest</li> <li>JavaProjectTest</li> <li>Graduation (Context)</li> <li>Graduation (Context)</li> <li>MainClass, Java</li> <li>MainClasstfi Java</li> </ul>	Source fol <u>d</u> er: Pac <u>k</u> age: Orch. Class	Element Location javaProjectTest/src java.project.test MainClass Element Description		Br <u>o</u> wse Bro <u>w</u> se Bro <u>w</u> se	Location /opt/C	
<ul> <li>Composition of the project of the pro</li></ul>	Modifier Return Type Parameters	□ private □ static	Choose a Me	Method Selec	tion 🔀	escription
	Constraints	Name Value Service Interface		Cancel	ок	× = -
< m •	?		Cancel	Einish		

Figure 32. Snapshot for selecting an existing method as Orchestration Element.

## **3.2. Deployment**

The COMPSs IDE allows developers to easily deploy and run applications in different infrastructures. To achieve it, developers have to select the deployment option, fill the configurations fields for the selected option and click "Deploy".

3.2.1. Deployment Options

# Deployment in Private Grid

The COMPSs IDE allows developers to easily deploy and run the application across a set of hosts distributed through the internet. To perform it, developers have to select the "*Private Grid*" deployment type in the *Build and Deploy* tab of the *Application Editor*. After selecting this option, several expandable menus will appear to configure different aspects of the application deployment. These are going to describe in detail in next paragraphs

## **Resource Selection**

The first step a developer has to do is defining and selecting the resources of their private grid which are going to be used for this deployment. This information has to be introduced in the *Resource Selection* section as you can see in Figure 33.

e Application Editor 🕴	- 8
Deployment Overview	<u>^</u>
Deployment	
This section provides the interface to deploy the application in a distributed computing platform Deployment Type Private Grid	
□ Deployment Options	
<ul> <li>Resource Selection</li> <li>Select the resources to be used in the deployment from the available ones</li> </ul>	=
Grid Resources	
Import Create Update Delete	
/Master Resource	
✓	
r Worker Resources	
Reset Save	
<ul> <li>Resource Deployement Properties</li> <li>Shared Disks Definition</li> </ul>	
Implementation Build and Deploy metadata.xml project.xml resources.xml	-

Figure 33. Snapshot of the Resource Selection part for the Private Grid deployment.

The resources definition can be imported from a file or create it from scratch. For the former case, developers have to click *"Import..."* and select the xml file which contains the resource definition with the COMPSs resources format. For the latter case, developers have to click *"Create..."* for creating a new resource definition. As result of this click, a dialog like Figure 34 will be open, where the developer can introduce the resource details. The *Resource* field will be used by the runtime to connect to this host. So, it must contain the either the *Fully Qualified Domain Name* of the resource or it IP address.

Resource
Hardware Properties
Processor Architecture
Processor Speed
Processor Count
Memory Size
Storage Size
Hardware Properties
Operating System
Installed Software Add
Host Properties
Task Count
Available Queues Available Queues
Cancel OK

Figure 34. Snapshot of the Grid Resource creation dialog.

Developers can also update existing resource definition by clicking *"Update..."* and selecting the resource to update in the dialog window; or deleting resource definitions by clicking *"Delete..."* and selecting the resource to delete in the dialog window.

The most important thing to define in this step is selecting which of the defined resources are going to be used as master or worker nodes. On one hand, the master node is defined by choosing a node from the *Master Resource* combo box. On the other hand, worker nodes are defined by passing the desired resources from the list on the left to the list on the right. To pass a resource the developer has to select the resource in the list and click ">".

# Define Deployment Information

The second step for deploying the application in a private cloud is to define the information to access and install the application in the selected resources. Developers can define this information in the *Resource Deployment Properties* section (Figure 35). For each resource, the developer has to define the username, the folder to install the application, the folder used by the runtime as working directory (a folder to store renamed and temporal files), and in case of the application contains web services, the location of the application server container.



The IDE, like COMPSs, uses SSH DSA keys to connect nodes. It is important that all the users used in each Grid nodes have properly installed the same key in the *authorized\_keys* file located in *\$HOME/.ssh* 

Deployment Optio	ns	
Resource Sele	ction	
<ul> <li>Resource Depl</li> <li>Describe the resource</li> </ul>	oyement Properties rce properties for deploying the application.	
Resource	✓	
Username		
Install Folder		
Working Folder		
App. Server Fold	er	
Reset Save		

Figure 35. Snapshot of the Resource Deployment Properties section for a private Grid deployment

### Define Shared Disks

In the case of a shared disk space is available on the Private Grid, it can be defined in the *Shared Disks Definition* section (Figure 36). There, developers can introduce a share disk definition by clicking *"Create..."* and introducing a name for identifying the shared disk and the mount point for each resource that can access to this shared disk.

ees Application Editor Σ	3	- 8			
Deployment Over	view				
Deployment This section provides t	the interface to deploy the application in a distributed computing platform				
Deployment Type	Private Grid				
⊡ Deployment Opti ▶ Resource Sel					
✓ Shared Disks	<b>loyement Properties</b> Definition g shared disks between the selected resources.	=			
Shared Disk	Create) Delete				
Master mount p	ath Resource Mount path				
Resources mou	Add				
Reset Save					
▶ Data Staging					
Deploy		~			
Implementation Build a	nd Deploy metadata.xml project.xml resources.xml				

Figure 36. Snapshot of the Shared Disk definition section for a Private Grid deployment.

# Define Data Staging

Another optional step is the *Data Staging*. If the execution of your application require input files or generate output files, the IDE can upload and download them from the master node before and after the application execution. These files have to be defined in the *Data Staging* section (Figure 37) by defining the path of the *Source File* and the *Destination Folder*. In the stage-in case, the *Source File* must include the path and file name in the local host and the *Destination Folder* should be the location in the remote master node. In the stage-out, the *Source File* must include the path and file name in the remote and the *Destination Folder* should be the location in the remote master node and the *Destination Folder* should be the location in the remote master node and the *Destination Folder* should be the location in the remote master node and the *Destination Folder* should be the location in the remote master node and the *Destination Folder* should be the location in the remote master node and the *Destination Folder* should be the location in the remote master node and the *Destination Folder* should be the location in the remote master node and the *Destination Folder* should be the location in the remote master node and the *Destination Folder* should be the location in the local host.

e Application Editor	3	- 8
Deployment Over	view	$\widehat{}$
Deployment This section provides t	the interface to deploy the application in a distributed computing platform	
Deployment Type	Private Grid	
🖃 Deployment Opti	ons	
Resource Sel	ection	
	loyement Properties	=
<ul> <li>Shared Disks</li> <li>Data Staging</li> <li>Specify the applic</li> </ul>	Definition	
Stage-in Files	Source File Destination Folder Add Delete	
Stage-out Files	Source File Destination Folder Add Delete	
Deploy		>
Implementation Build a	and Deploy metadata.xml project.xml resources.xml	

Figure 37. Snapshot of the Data Staging section for a Private Grid deployment.

Once the configuration has finished, developers can start the application deployment by clicking *"Deploy"*.



The IDE uses the Eclipse Ant plugin with JSCH library to make copy and execute command remotely. The JSCH library (*jsch-0.1.xx.jar*) is automatically copied to the *\$HOME/.ant/lib* folder. However, some Eclipse installations do not properly detect the jars installed in this folder. If the deployment fails because Ant does not find JSCH classes, you must manually configure the Ant Plugin to use the JSCH libray. It can be done by adding this jar (*\$HOME/.ant/lib/jsch-0.1.xx.jar*) as a Global Entry of the Ant Runtime configuration. This configuration can be found in menu *Window->Preferences* and once in the Preferences page select *Ant->Runtime* 

#### Deployment with the Enactment Service

The COMPSs IDE allows developers to easily deploy and run the application in cloud environments through the COMPSs Enactment Service. To perform it, developers have to select the *"Enactment Service"* deployment type in the *Build and Deploy* tab of the *Application Editor*. After selecting this option, several expandable menus will appear to configure different aspects of the application deployment. These are going to describe in detail in next paragraphs:

#### Define Storage Service

The *Storage Service* section (Figure 38) is used to define the details of the Storage Service used to manage the application files in a remote storage. This section is divided in three parts: the *Credentials* part, the *Location* part and the *Application* part.

In the *Credentials* part, developers just have to specify the username and password to access the storage service. The *Location* part is used to define the endpoint reference of the Storage Service, defining the hostname, protocol and port (if it is not standard). Finally, developers can configure the details of the application location inside the storage service in the *Application* part.

e Application Editor 🕴	- 8
Deployment	^
This section provides the interface to deploy the application in a distributed computing platform	
Deployment Type Enactment Service	
□ Deployment Options	
<ul> <li>Storage Service</li> <li>Define the storage service for uploading the application data.</li> </ul>	
Credentials	
Username	
Password	
	=
Protocol Port	
Hostname	
Application	
Application Folder	
Stage-in Folder	
Source File Destination Folder Add	
Stage-in Files	
Stage-out Folder	
File	
Stage-out Files	
Delete	
Logs Folder	
Reset	
Implementation Build and Deploy metadata.xml project.xml resources.xml	~
npementation Durid and Deploy Interadata.xnn project.xnn resources.xnn	

Figure 38. Snapshot of the Storage Service section for a deployment with the Enactment Service.

More in detail, the *Application Folder*, contains the folder where the application package will be uploaded in the Storage Service. The *Stage-in*, *Stage-out* and *Logs Folders* define the folder where the application package, stage-in, stage-out and log files will be stored in the Storage Service. Finally, the *Stage-in* and *Stage-out Files* define the data staging required to run the application. In the stage-in case, the *Source File* field must contain the path and filename of the stage-in files in the local host, and the *Destination Folder* must contain the path to store this file in the master Cloud VMs, while, in the stage-out case, developers only need to specify the path of application output files in the master Cloud VMs. These files will be automatically uploaded to the Storage Service in the location indicated by the *Stage-out Folder* once the application run finishes.

## Define Enactment Service

The *Enactment Service* section (Figure 39) is used to define the details of the service used to run the application in a Private Cloud. As in the *Storage Service* section, it is also divided in three parts: the *Credentials* part, the *Location* part and the *Application* part.

In the *Credentials* part, developers have to specify the username and password to access the Enactment Service. The *Location* part is used to define the endpoint reference of the Enactment Service. Finally, developers can configure application details like the VM image, the maximum and minimum number of VMs and the wall-clock-limit in the *Application* part.

🧙 Application Editor 🕴				
Deployment Overview				
Deployment				
This section provides the interface to deploy the app	lication in a distributed computing platform			
Deployment Type Enactment Service	Deployment Type Enactment Service  ≎			
Deployment Options				
Storage Service				
<ul> <li>Enactment Service</li> <li>Provide the configuration of the enactment serving application</li> </ul>	ce for deploying the			
Credentials				
Username				
Password				
Location				
Service Endpoint				
Application				
VM Image				
Maximum VMs				
Minimum VMs				
Wall Clock Limit				
Deploy				
Implementation Build and Deploy metadata.xml pro	iect.xml resources.xml			

Figure 39. Snapshot of the Enactment Service section for a deployment with the Enactment Service.

Once the configuration has finished, developers can start the application deployment by clicking *"Deploy"*.



The current version of the Enactment Service only supports the execution of applications with a single Orchestration Element which is the main method of the application. If you try to deploy and run other type of applications with this deployment option you will get an error message during the deployment process alerting about it.



Calling the Enactment Service require importing the Enactment Service credential on the JRE KeyStore.

keytool -import -file bscpmes.cer -alias pmes -keystore \$JAVA\_HOME/jre/lib/security/cacerts

If you are running Eclipse with the Java 6, you should add the WSIT libraries in the java endorsed directory and define it in the JVM options of the Eclipse launch script

-Djava.endorsed.dirs=<path\_to\_libraries\_dir>

# 3.2.2. Select the Main Class

Once deployment process has been started, the IDE performs the compilation, instrumentation and packaging of the application and the deployment in the selected distributed infrastructure. If during this process the IDE detects an application is not a Web Service, the IDE will ask the developer if she/he want to select the class which contains the application main method (Figure 40).

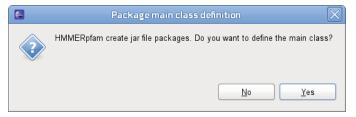


Figure 40. Snapshot of the main class warning.

If click yes, the IDE will open a window, as Figure 41, to select where it can find the main class. The process is the same as for selecting existing classes and it was explained in sections 3.1.7 and 3.1.8. If the main class is an imported class, the developer has to specify the package location and select a class from the package otherwise she/he just has to select a class.

	Application Editor 😫									
υ		X								
Ξ	Select the class which contains the main method.									
	Select the class which contains the main method. It must be executed when running the application. The class could be selected from existing classes (leave package and sub-package location empty)									
	Package Location		Select							
C	Sub-package (if any jar)		Select							
	Orch. Class		Select							
Imple	(?)	Choose a class from the list  Choose a class from the list  Addening items:  Source  Cancel  C	Einish							

**Figure 41. Snapshots for selecting the main class** 

#### 3.2.3. Application Status View

Once the deployment has finished, the IDE shows the state of the application deployment in the *Application Status* view. In general, this view (Figure 42) shows the application *identifier*, the *status* of the deployment and a combo and a table to show where the different parts of the application has been deployed. Additionally, it also contains the *"Updeploy"* button and the *Keep Data* checkbox for un-deploying the application with option of removing or not the service data.

😰 Problems 🙋 Javadoc 😟 Declaration 🥽 Application Status 🕱				
∽ GeneDetectionBroker				
Deployment Management				X
Identifier: GeneDetectionBroker	Status: De	ployed	Undeploy 🗌 Keep Data	
Provider: Private Grid	Instances:	Address Type		
		cores_1 bscgrid06.bsc.es		
		cores_0 bscgrid05.bsc.es		
		master bscgrid05.bsc.es		

Figure 42. Snapshot of the Application Status view for a service deployment.

For web service applications, the IDE also starts the application server during the deployment process. The user do not need to do anything more to start the application, just invoke the service. In the case of JAR application, the IDE only copy the application files during the deployment process. Therefore, the *Application Status* view includes the "Start Execution" and "Cancel Execution" in this case (Figure 43). These buttons allow developer to start and cancel the application execution in the distributed infrastructure.

😨 Problems 🔎 Javadoc 😥 Declaration 🤤 Application	n Status 🖾		- 0
✓ HMMERpfam			
Deployment Management			X
Identifier: HMMERpfam Status: Deployed Cancel Execution Start Execution Undeploy			
Provider: Private Grid	Instances:	Address   Type	
		cores_1 bscgrid06.bsc.es	
		cores_0 bscgrid05.bsc.es	
		master bscgrid05.bsc.es	

Figure 43. Snapshot of the Application Status view for a Jar application deployment.

## **Known Limitations**

This version of the COMPSs IDE contains a set of known limitations. They will try to be solved in next releases.

Current version of the IDE does not support deleting Orchestration and Core elements from the service editor. You must do it by hand deleting the annotations in the Orchestration class or the Core Element Interface.

The Enactment Service deployment is limited to the type of applications supported by the Enactment Service tool. They are JAR applications created from scratch with a default main Orchestration Element.

# References

- [1] Eclipse Platform web site. <u>https://www.eclipse.org/</u>
- [2] Eclipse Marketplace web site. <u>http://marketplace.eclipse.org/</u>
- [3] Oracle VM Virtual Box web site. <u>https://www.virtualbox.org/</u>
- [4] COMP Superscalar web site. <u>http://compss.bsc.es</u>/
- [5] Apache Subversion web site. <u>http://subversion.apache.org/</u>
- [6] Apache Maven web site. <u>http://maven.apache.org/</u>