

Advanced Build System

ABS - Reference Documentation

Release 0.7

Table of Contents

Preface	iv
1. ABS basics	1
1.1. What is a component ?	1
1.2. What is a dependancy ?	2
1.2.1. Binary dependency	2
1.2.2. Source dependency	2
1.3. What's a project ?	3
1.4. What's a deployment	3
1.5. What's a module ?	3
1.6. What's a task / activity ?	4
2. Installation	5
2.1. ABS basic installation	5
2.2. ABS Eclipse plugin installation	6
3. Command line usage	9
3.1. what's a workspace ?	10
3.2. Project creation	11
3.2.1. Create a project from scratch	11
3.2.2. Retrieve an existing project	12
3.2.3. Retrieve an existing project from a specific version	12
3.3. Component creation	12
3.3.1. Create a new component	13
3.3.2. Retrieve an existing component	13
3.3.3. Retrieve an existing component from a specific version	13
3.4. Compiling	14
3.5. Create an new deployment target	14
3.6. Delivering application	15
4. Using ABS with Eclipse	16
4.1. Creating an new component	16
4.2. Creating a new project	19
4.3. Creating a design model	22
4.3.1. UML modelization	22
4.3.2. Logical architecture	24
4.3.2.1. Stereotypes	24
4.3.2.2. <<Entity>> stereotype	25
4.3.2.3. <<Dto>> stereotype	26
4.3.2.4. <<EntitiesManager>> stereotype	26
4.3.2.5. <<Process>> stereotype	26
4.3.2.6. <<Controller>> stereotype	27
4.3.2.7. <<Ui>> stereotype	28
4.3.2.8. <<View>> stereotype	28
4.3.2.9. <<Remote>> stereotype	28
4.3.2.10. <<Transactional>> stereotype	29
4.3.2.11. <<Config>> stereotype	29
4.4. Generating code	29

4.4.1. Spring based physical architecture	36
4.4.1.1. Prerequisite	36
4.4.1.2. Generated files	38
4.5. Testing generated code	40
4.5.1. technical layers configuration	40
4.5.1.1. component.xml file	40
4.5.1.2. component-test.xml file	41
4.5.1.3. applicationContext-tests.xml file	41
4.5.2. Testing data access layer	42
4.5.3. Testing business process layer	44
4.5.4. Testing Web Services	44
4.6. Running a project inside Tomcat	44
4.6.1. Configure your project for webapps deployment. déployer en tant qu'application Web	44
4.6.1.1. web.xml file	44
4.6.1.2. Database access configuration	45
4.6.2. Run Tomcat inside Eclipse	46

Preface

ABS means Advanced Build System, it's a component oriented toolchain.

It's not a Maven clone. it has some advanced components assembly features. ABS focus on simplicity and pragmatism, its first objectives are automatization, normalization and orchestration of computer applications build cycle. It enforces the MDA way in design activity.

ABS integrate several free java tools and offert a simple unique environment for all activities described in our component oriented methodology. All those tasks may be triggered using command line ant script or using Eclipse plugin.

ABS building environment offer a simple and pragmatic way to build component oriented MDA project. The differents available features are :

- Component or project creation
- Documentation generation based on UML model content (using PragMatic and Acceleo technologies)
- Managing external binary and internal source dependencies
- Integration with repository management system like CVS and Subversion
- sources compilation
- Unit tests run
- Code generation
- Continuous integration
- Deployment management
- and loads of others things ...

The ABS project is a free and collaborative project. You'll find more details on the web site [<http://www.sharengo.org/Wiki?ABS>] : latest versions download, tutorials, bugtracker and mailing lists.

1

ABS basics

As we have seen in introduction, ABS main ambition is to industrialize software production. The first we took, was to define a metamodel used to define all the concepts related to the project lifecycle : project, component, dependencies, deployment, ... In the following paragraphs, we will define all those concepts.

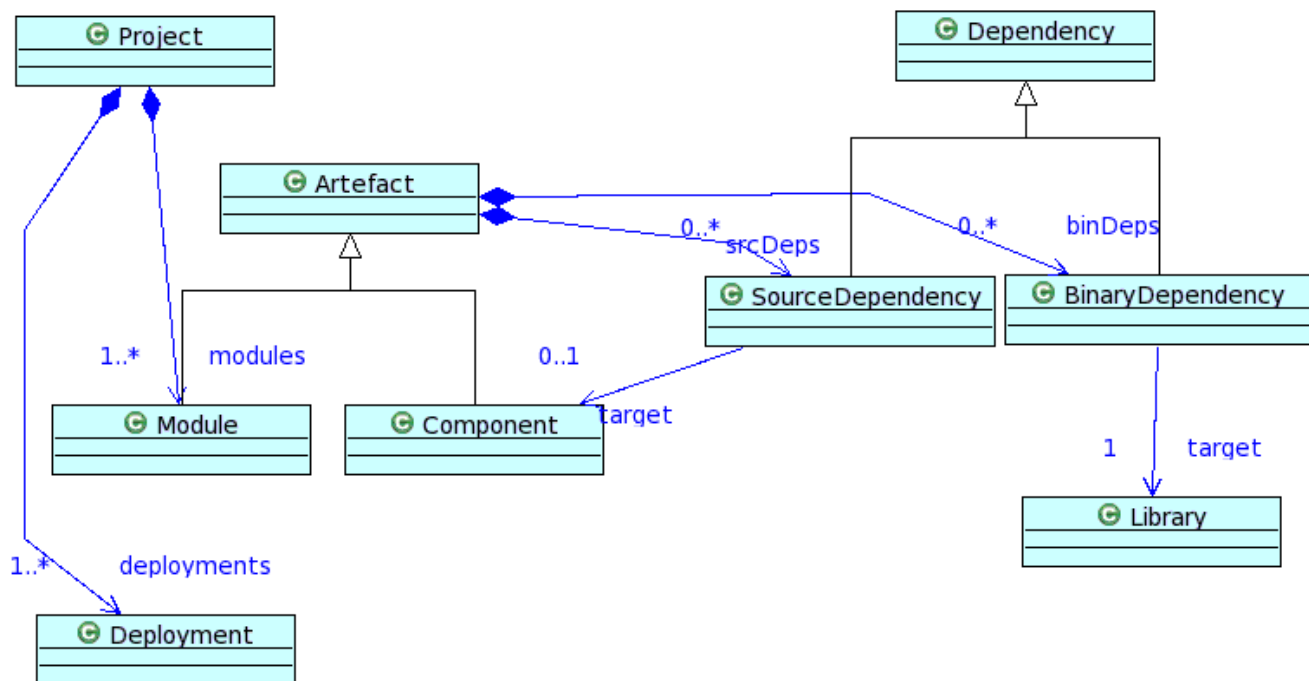


Figure 1.1. Métamodèle ABS

1.1. What is a component ?

Component Oriented Development is a way of building software using a modular architecture resulting in better understanding of system behaviour, easier maintenance and high level of reusability. In our context a component is :

- a coherent business entity. For example : Customer, Bill, Order, ..
- defined through a model
- publishing business services reusable in different contexts

- containing java source code, configuration files, all generated from its descriptive model. All those files defined the reference implementation. It's possible to use the same model to generated source code related to another target technology.

1.2. What is a dependancy ?

1.2.1. Binary dependency

A binary dependency define a usage relationship between a component and an external binary library (build outside ABS). A project module (define later) may use this kind of dependency too. All component or module binaries dependencies are stored in the `bindep.txt` file located at the root of the component directory.

Example 1.1. bindep.txt file example

```
commons-logging.jar
spring.jar
-servlet-api.jar.
```

Those dependencies are mainly used during compilation and to build application package to deploy. Librairies are automatically downloaded in directory `$ABS_HOME/binary-repository` if not present on the local workstation just before compilation.

ABS knows where to look for missing librairies and where to store them after download using properties in `build.properties` file located in `$ABS_HOME`.

1.2.2. Source dependency

A source dependency define a usage relationship between two components : a A component needs a B component in order to run properly.A project module may used this kind of relationship with components. Source dependencies are stored in a `srcdep.txt` file located at the root of the component or module directory.

Example 1.2. srcdep.txt file example

```
bc:componentA
sharengo:domainB/componentC/trunk
```

The previous example shows two dependencies declaration.Format of source dependencies are : `scm_repository:target:`

- `scm_repository` : logical name of cvs or svn repository. These name is defined in file `$ABS_HOME/repositories.properties` cf. section TODO
- `:` : séparateur
- `ltarget` : component name. The second line in the example reference a component store in a svn repository and

express directly dependency versioning, that is the current component needs trunk version of C component.

1.3. What's a project ?

A project is an application, whose delivery is the final objective. it's the result of several business components assembly. It defines target running environment named "deployment" which may have different and specific configuration.

1.4. What's a deployment

A deployment is a target running environment receiving war, tar.gz and jnlp files. We can define several different deployments in the same project, for example :development, customer test, production, ... Deployments list is stored in file `deployment.txt` located in the root directory of the project.

Example 1.3. `deployment.txt` file example

```
localhost
test
prod
```

The previous example shows 3 deployments. Very often each environment has distinct configuration properties. For example SMTP server to use is different for the development team and at the customer production site. We have to parametrize this information using a replacement token.

Example 1.4. Properties file containing SMTP server address definition

```
smtp.server.name=@smtp.server.name@
```

Example 1.5. `deployment/test/replace.server.properties` file describing deployment parameter for value for test environment

```
@smtp.server.name@=test.mail.sharengo.org
```

Example 1.6. `deployment/preprod/replace.server.properties` describing production environment

```
@smtp.server.name@=mail.sharengo.org
```

The two previous files allow to have different smtp server for each environment.

1.5. What's a module ?

The module concept is equivalent to a component but it is not reusable and therefore related only to one project. Very often, modules are only used to defined project dependencies. We can find very specific source code needed for migration or stuff like that. Modules list is stored in `moddep.txt` file located at project root directory.

Example 1.7. moddep.txt file example

```
main
```

This example shows a declaration for a module call "main" inside a project.

1.6. What's a task / activity ?

We have seen previously that ABS define tasks related to project methodology. You start by creating a project using `abs new.project` command and specifying a name. For each following task execution, ABS will load the project / component / module definition target of your command and will inject it in a velocity template containing a description of an Ant script to instanticiate. A script Ant is generated and run. One advantage of this approach is to completely separate project description from build tools. If you want to change the scripting langage and use something else, do it, that's easy. You have to see ABS as a logical "shell" running abstract project command in a project context. Moreover it's very configurable and use a package manager to upgrade easily developper workstation and so doing to provide new command very quickly. Developpers doesn't have to modify build script anymore.

2

Installation

2.1. ABS basic installation

ABS needs java version 5 or greater.

- Create a directory for ABS (for example ~/abs).
- Download `installer` (workshop-repository-manager-0.6.jar [<http://prdownloads.sourceforge.net/abs/workshop-repository-manager-0.6.jar?download>])
- Start installation from install directory (~/abs), running the following commands :

```
java -jar workshop-repository-manager-0.6.jar http://sharengo.org/abs/0.7/abs-core
java -jar workshop-repository-manager-0.6.jar http://sharengo.org/abs/0.7/mda
java -jar workshop-repository-manager-0.6.jar http://sharengo.org/abs/0.7/qa
java -jar workshop-repository-manager-0.6.jar http://sharengo.org/abs/0.7/doc
```

If you have to use a proxy, Si, vous souhaitez utiliser un proxy, you have to use the following syntax : `java -jar workshop-repository-manager-0.6.jar repository-url [proxy port [user pwd]]`

- Define `ABS_HOME` variable in your environment (`ABS_HOME=~/abs`)
- Add directory `$ABS_HOME/tc/ant/bin` in your system path
- Initialize toolchain

TODO vérifier que cette commande existe encore, si oui la supprimer et créer un fichier `build.properties` par défaut.

```
cd $ABS_HOME
abs init
```

Answer the script questions (some sensible default values are provide, use it if you don't know the answer). This initialization phase will create a `build.properties` file in your `$ABS_HOME` directory. Here is a `build.properties` example :

```
binary.server.url=http://adk.open-model.org/adk-2.2/binary-components
binary-repository.path=/home/jeromeb/dev/abs/binary-repository
jvm.target=1.5

ci.server=build.argia.fr
ci.user=XXXX
ci.password=XXXX
```

2.2. ABS Eclipse plugin installation

Upload Eclipse version 3.3 (Europa) for your system : here [<http://sharengo.org/update/europa>] .Start eclipse, select menu option "Help" -> "Software Updates" -> "Find and Install ..." :

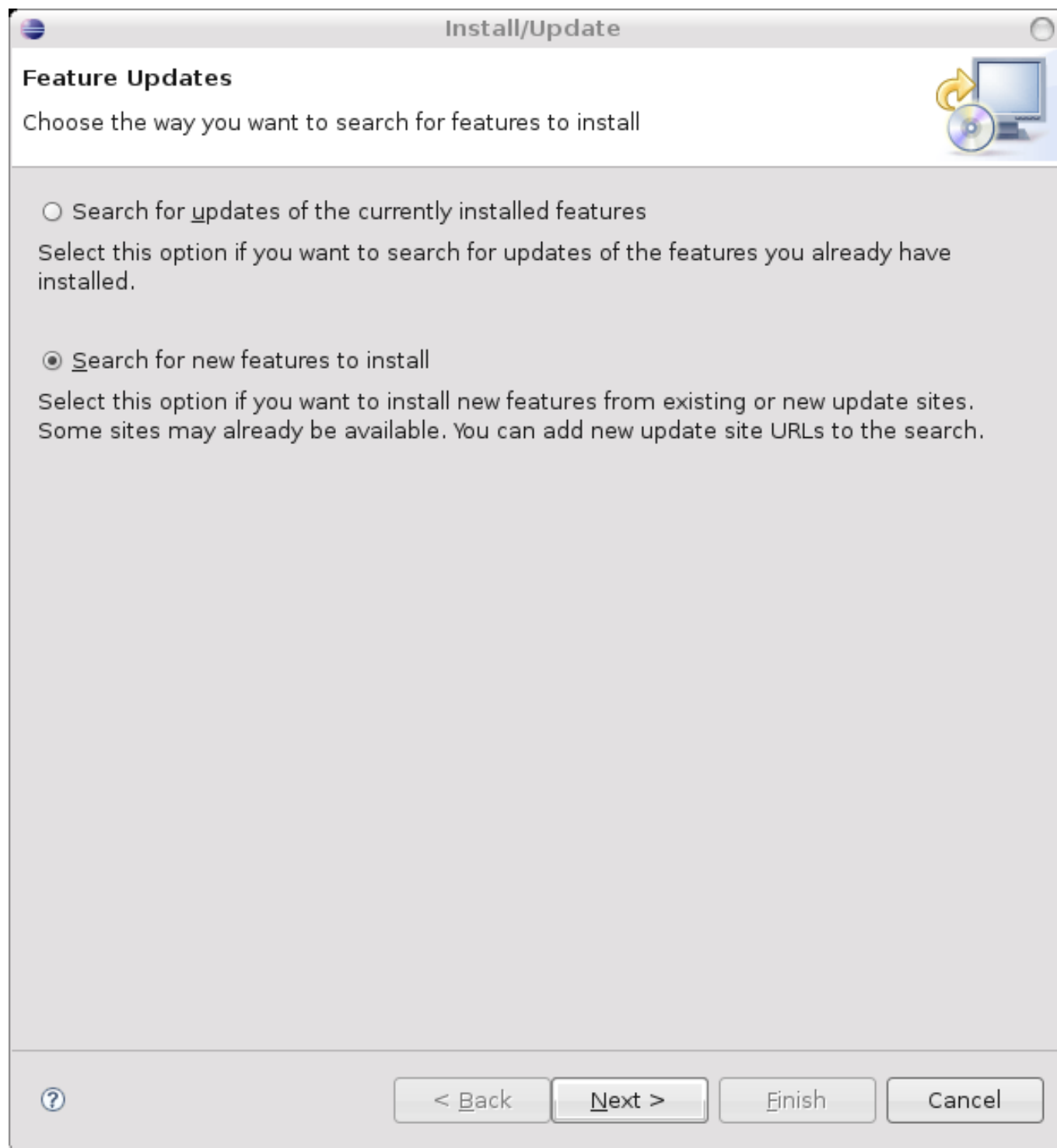


Figure 2.1. Software updates screen shot

Choose "Search for news features to install" and click "Next" :

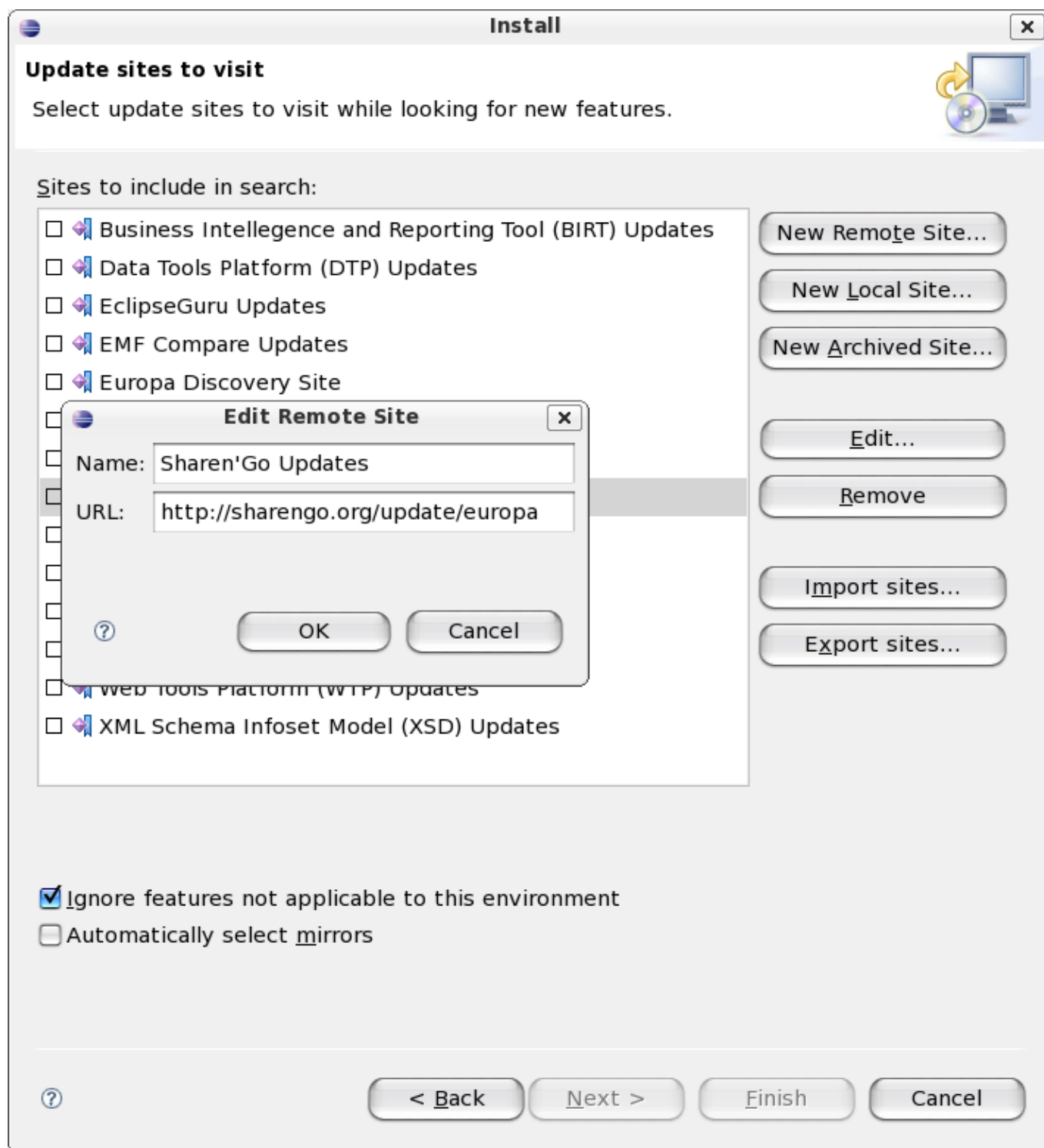


Figure 2.2. Update sites screen shot

Using "New Remote Site" button, add Sharen'Go update site : <http://sharengo.org/update/europa>, select it and click "Finish" Select Sharen'Go tools in the available features list and click "Next" :

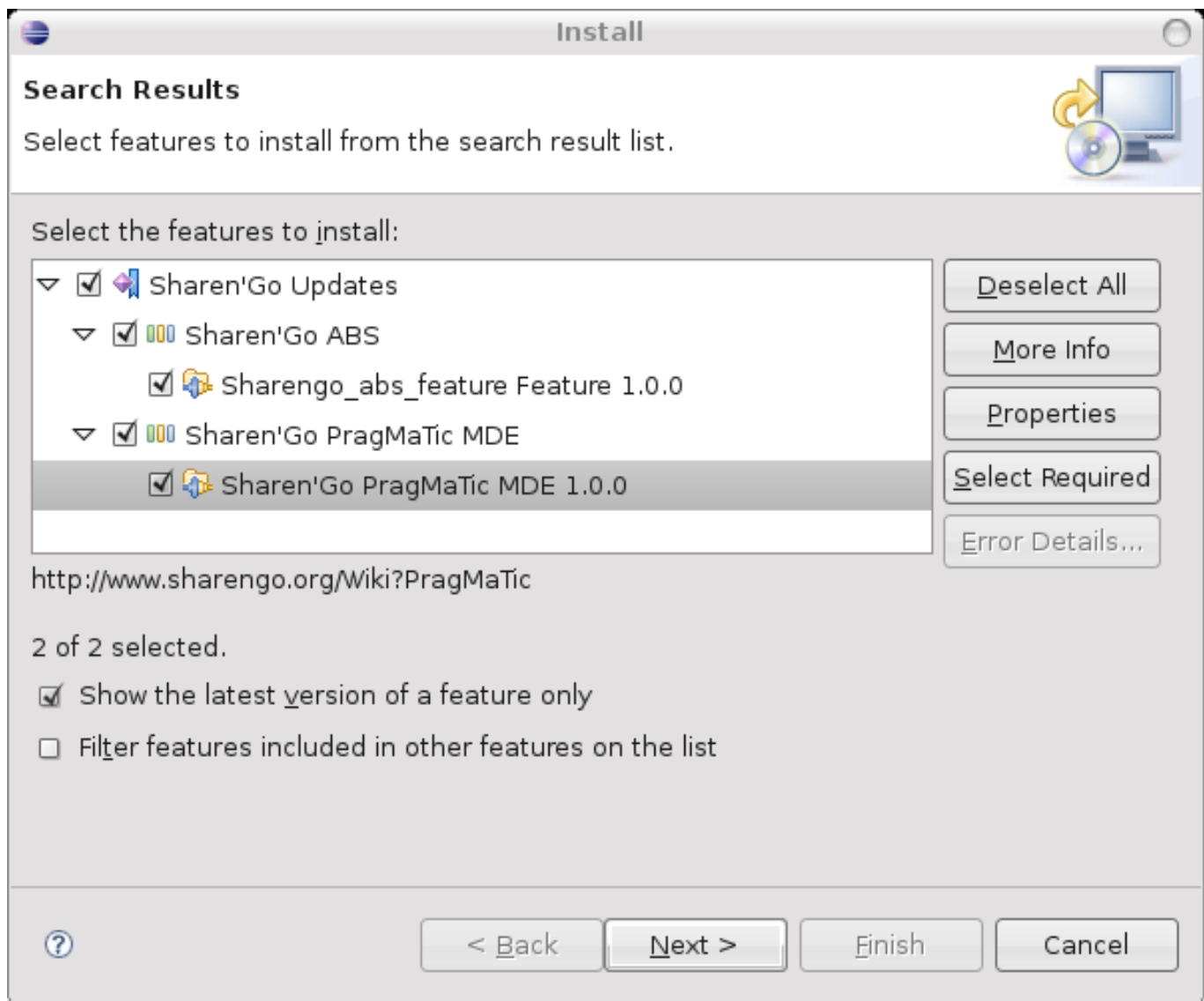


Figure 2.3. Install features screen shot

Accept license and click "Finish"

3

Command line usage

This chapter doesn't not contains detailed informations, it presents the main useful commands. A more complete list can be obtained with the following command :

```
$> abs -p
```

Output example :

```
$> cd $ABS_HOME
$> abs -p
new.workspace      Create new workspace in absolute directory
new.component      Add new component in production
new.project        Create new project in production
new.scm.component  Get an existing component in production
new.scm.project    Get an existing project in production

$> cd /home/jeromeb/workspaces/ws-workspace1/projects/project1
$> abs -p
abs:bug.report     Report bug in tool chain
abs:help           Show Online Help, like man page.
abs:upgrade.build  Upgrade build file
abs:version        Show ABS version
checkstyle:all    Run checkstyle audit on local and dependent source code
checkstyle:help   Show Online Help, like man page.
checkstyle:local  Run checkstyle audit on local source code
ci:register        Register project in continuous integration server
classycle:all     Execution de l'analyse classycle
classycle:clean   Suppression des xml generes par classycle.run
classycle:local   Execution de l'analyse classycle
clean:all         remove all local and dependent builded files
clean:help       Show Online Help, like man page.
clean:local      remove all local builded files
dep:add.bin       Add a new dependency to a binary component
dep:add.mod       Add a new module to a project
dep:add.src       Add a new dependency to a source component
dep:help         Show Online Help, like man page.
dep:ls.bin       List all binary dependencies
dep:ls.mod       List all modules
dep:ls.src       List all source dependencies
dep:mv.bin       Change binary dependencies order
dep:mv.mod       Change module dependency order
dep:mv.src       Change source dependencies order
dep:rm.bin       Remove one binary dependency
dep:rm.mod       Remove module
dep:rm.src       Remove one source dependency
deploy:build.cli  build all Command Line Interface for each deployment
deploy:build.client.cli  build all Command Line Interface for each deployment. (just include common and client code)
deploy:build.client.jnlp  build all jnlp for each deployment. (just include common and client code)
deploy:build.war  build all Webapps Archive for each deployment
deploy:help      Show Online Help, like man page.
deploy:new       create new deployment target
deploy:rm        remove deployment target
doc:all.design.pdf  Generate SVG diagrams, DocBook documentation stored in UML model and fir
doc:analysis.pdf  Generate SVG diagrams, DocBook documentation stored in UML model and fir
```

doc:client.design.pdf	Generate SVG diagrams, DocBook documentation stored in UML model and fir
doc:common.design.pdf	Generate SVG diagrams, DocBook documentation stored in UML model and fir
doc:help	Show Online Help, like man page.
doc:needs.pdf	Generate SVG diagrams, DocBook documentation stored in UML model and fir
doc:server.design.pdf	Generate SVG diagrams, DocBook documentation stored in UML model and fir
eclipse:gen.classpath	Generate classpath for eclipse project (all dependents components are lo
eclipse:gen.classpath.full.src	Generate classpath for eclipse project (all dependents components are lo
eclipse:gen.project	Generate required resources for eclipse project
eclipse:help	Show Online Help, like man page.
hibernate:help	Show Online Help, like man page.
hibernate:schemaexport.all	generate SQL source code and dependent component from Hibernate mapping
hibernate:schemaexport.local	generate SQL source code from Hibernate mapping description
javadoc:all	generate javadoc for local and dependent generated deliverables
javadoc:help	Show Online Help, like man page.
javadoc:local	javadoc for all local java files
lib:all	Compile all local and dependent components
lib:aspectj.all	Compile all local and dependent components with AspectJ compiler in orde
lib:aspectj.local	Compile local files with AspectJ compiler in order to support OAP.
lib:help	Show Online Help, like man page.
lib:local	Compile local files
mda:apply.patch.local	apply all patches on local generated file by others mda tasks
mda:gen.hbm.local	generate local hibernate mapping file from UML models
mda:gen.java.local	generate local java source code from UML models
mda:help	Show Online Help, like man page.
mda:pim2pim.local	Provide model-to-model transformation in order to check coherency of Pla
netbeans:deploy.war	deploy application in netbeans tomcat instance
netbeans:gen.all.project	Generate required resources for netbeans project
netbeans:gen.local.project	Generate required resources for netbeans project
netbeans:help	Show Online Help, like man page.
netbeans:undeploy.war	undeploy application in netbeans tomcat instance
pmd:all	Audit source files using company conventions
pmd:help	Show Online Help, like man page.
pmd:local	Audit local java files using company conventions
pmd:report.all	build audit source files using company conventions
pmd:report.local	Build audit local java files using company conventions
scm:changelog.all	build changelog for local files and dependent components
scm:changelog.local	build changelog for local files
scm:co.all	check out local files and dependent components
scm:co.local	checkout local files
scm:help	Show Online Help, like man page.
scm:initial.import	first import of component directory in scm repository
scm>manual.tag.all	Set to all source files and dependent component from scm repository a ne
scm:prepare.tree.all	add empty dir to each dependent component after checkout
scm:prepare.tree.local	add empty dir to current component after checkout
scm:tag.all	Set to all source files and dependent component from scm repository a ne
scm:tagdiff.all	build changelog for local files and dependent components
scm:tagdiff.local	build changelog for local files
scm:update.all	Update source files and dependent component (preserve sticky tags)
scm:update.local	Update source files (preserve sticky tags)
scm:update.to.tag.all	Update source files and dependent component with specifieg tag
scm:update.to.tag.local	Update source files with specifieg tag
scm:update.to.trunk.all	Update source files and dependent component and reset sticky tags
scm:update.to.trunk.local	Update source files and reset sticky tags
srcformat:all	Reformat source files using company conventions
srcformat:help	Show Online Help, like man page.
srcformat:local	format local java files using company conventions
test:help	Show Online Help, like man page.
test:java.all	Run local Unit tests local and dependent components
test:java.local	Run local Unit tests

3.1. what's a workspace ?

A workspace permits to separate working context. It defines a physical storage area for a project and all the needed components. A good practice is to keep a distinct workspace for each version of a project you work on. So when you are working on a given project phase, you have to create a workspace. Then when you'll start a new phase, you'll create a new workspace and so doing isolate phase2 from phase 1. If you want to do some modifications on the first one, you'll be able to deliver a new version with only your small modifications without any evolution of phase 2. You'll find thereafter a workspace creation command :

```
$> cd $ABS_HOME
$> abs new.workspace

abs:version:
  [echo]
  [echo] ~~~~~~
  [echo]   ABS ~ Advanced Build System ~
  [echo]
  [echo]           ~ Next Generation Build System ...
  [echo] ~~~~~~
  [echo]

new.workspace:
  [input] Enter workspace absolute path :
  /home/jeromeb/workspaces/ws-workspace1

...

BUILD SUCCESSFUL
```

3.2. Project creation

First, you need to position yourself in the directory related to the newly created workspace.

3.2.1. Create a project from scratch

The "new.project" command create a new local project (local means with no equivalent in the repository).

```
$> cd /home/jeromeb/workspaces/ws-workspace1
$> abs new.project

abs:version:
  [echo]
  [echo] ~~~~~~
  [echo]   ABS ~ Advanced Build System ~
  [echo]
  [echo]           ~ Next Generation Build System ...
  [echo] ~~~~~~
  [echo]

new.project:
  [input] Enter project name :
  project1

...

BUILD SUCCESSFUL
```

3.2.2. Retrieve an existing project

You can also locally retrieve a project store in a repository using the command "new.scm.project" (the configuration of the various accessible repositories are kept in `$ABS_HOME/repositories.properties` :

```
$> cd /home/jeromeb/workspaces/ws-workspace1
$> abs new.scm.project

abs:version:
  [echo]
  [echo] ~~~~~
  [echo]   ABS ~ Advanced Build System ~
  [echo]
  [echo]           ~ Next Generation Build System ...
  [echo] ~~~~~
  [echo]

new.scm.project:
  [input] Enter project name :
project2
  [input] Enter repository name :
sharengo

...
BUILD SUCCESSFUL
```

3.2.3. Retrieve an existing project from a specific version

The previous command has a variant to retrieve a project from a specific version. This command is useful mainly with cvs. If you use subversion, the previous command is sufficient because the version name is included in the project name, for example : `project1/tags/1.0`.

```
$> cd /home/jeromeb/workspaces/ws-workspace1
$> abs new.scm.project.from.tag

abs:version:
  [echo]
  [echo] ~~~~~
  [echo]   ABS ~ Advanced Build System ~
  [echo]
  [echo]           ~ Next Generation Build System ...
  [echo] ~~~~~
  [echo]

new.scm.project.from.tag:
  [input] Enter project name :
project3
  [input] Enter tag :
version1.0
  [input] Enter repository name :
sharengo

...
BUILD SUCCESSFUL
```

3.3. Component creation

First, you have to position yourself in the directory related to the current workspace.

3.3.1. Create a new component

The command "new.component" create a new local component (local means with no equivalent in the repository).

```
$> cd /home/jeromeb/workspaces/ws-workspace1
$> abs new.component

abs:version:
  [echo]
  [echo] ~~~~~
  [echo]   ABS ~ Advanced Build System ~
  [echo]
  [echo]       ~ Next Generation Build System ...
  [echo] ~~~~~
  [echo]

new.component:
  [input] Enter component name :
component1

...

BUILD SUCCESSFUL
```

3.3.2. Retrieve an existing component

You can retrieve an existing component stored in a repository using "new.scm.component" command.

```
$> cd /home/jeromeb/workspaces/ws-workspace1
$> abs new.scm.component

abs:version:
  [echo]
  [echo] ~~~~~
  [echo]   ABS ~ Advanced Build System ~
  [echo]
  [echo]       ~ Next Generation Build System ...
  [echo] ~~~~~
  [echo]

new.scm.component:
  [input] Enter component name :
component2
  [input] Enter repository name :
sharengo

...

BUILD SUCCESSFUL
```

3.3.3. Retrieve an existing component from a specific version

The previous command has a variant in which you can specify the target version of the component you want to get in the repository. This command is useful mainly with cvs. If you use subversion, the previous command is suffi-

cient because the version name is included in the component name, for example : component1/tags/1.0.

```
$> cd /home/jeromeb/workspaces/ws-workspace1
$> abs new.scm.component.from.tag

abs:version:
  [echo]
  [echo] ~~~~~~
  [echo]   ABS ~ Advanced Build System ~
  [echo]
  [echo]           ~ Next Generation Build System ...
  [echo] ~~~~~~
  [echo]

new.scm.component.from.tag:
  [input] Enter component name :
component3
  [input] Enter tag :
version1.0
  [input] Enter repository name :
sharengo

...
BUILD SUCCESSFUL
```

3.4. Compiling

The "lib.local" command is used to compile a component or a project. This operation build 3 jar archives containing all *.class files from the java source files. You can also use the command "lib.all" which apply the compilation process on all components used by the current one. The 3 resulting archives are :

- COMPONENT-NAME.jar : all class files from the "common" directory
- COMPONENT-NAME-client.jar : all class files from the "client" directory
- COMPONENT-NAME-server.jar : all class files from the "server" directory

```
$> cd /home/jeromeb/workspaces/ws-workspace1
$> cd components/component1
$> abs lib:all

...
BUILD SUCCESSFUL
```

3.5. Create an new deployment target

The "deploy.new" command create a new deployment environment for the current project. It add a directory with the deployment name inside the "deployment" directory located in project root directory. It also create *replace.server.properties* and *replace.client.properties* files containing the value of the variable part of the configuration.

```
$> cd /home/jeromeb/workspaces/ws-workspace1/projects/project1
$> abs deploy:new

abs:version:
  [echo]
  [echo] ~~~~~~
  [echo]   ABS ~ Advanced Build System ~
  [echo]
  [echo]           ~ Next Generation Build System ...
  [echo] ~~~~~~
  [echo]

deploy:new:
  [input] Enter deployment target name :
localhost

...
BUILD SUCCESSFUL
```

3.6. Delivering application

The "deploy:build.war" command build a WAR archive which can be deployed in an application server as Tomcat.

The "deploy:build.cli" command build a *.tar.gz archive containing an application environnement for command line interface program.

This 2 commands build one archive file for each deployment environment using the configuration parameter values found in replace files.

4

Using ABS with Eclipse

4.1. Creating an new component

In menu : "New" -> "Project" -> in ABS node select "New Component"

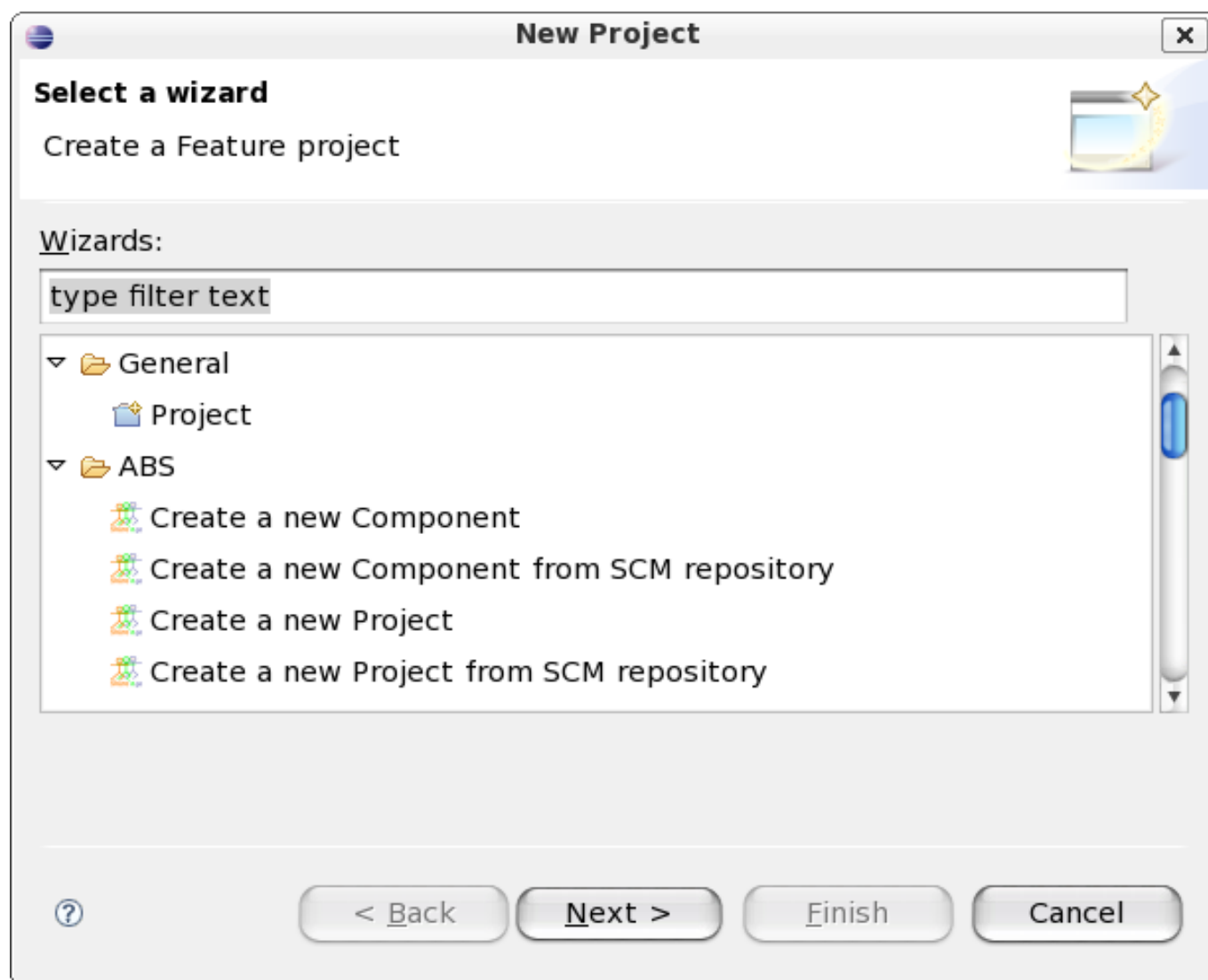


Figure 4.1. Screenshot from component creation

The eclipse plugin interacting with your locally installed ABS, it's mandatory to specify the installation directory `$ABS_HOME` and the ABS workspace. If those informations are not available in ABS eclipse configuration (menu

"Window" -> "Preferences ..." -> "ABS Preference"), the following wizard will be presented and will help you to configure :

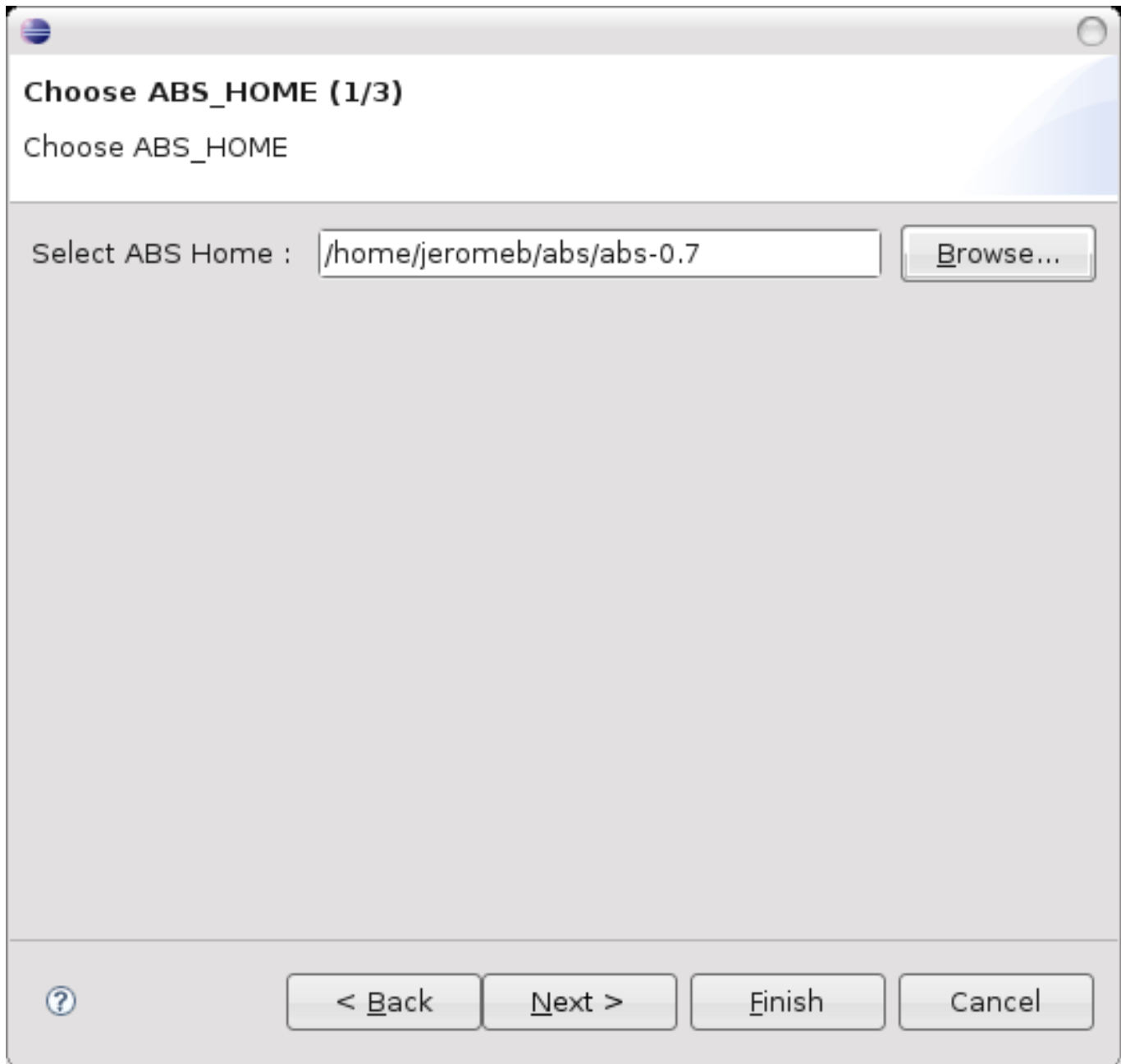


Figure 4.2. Screenshot for ABS_HOME selection

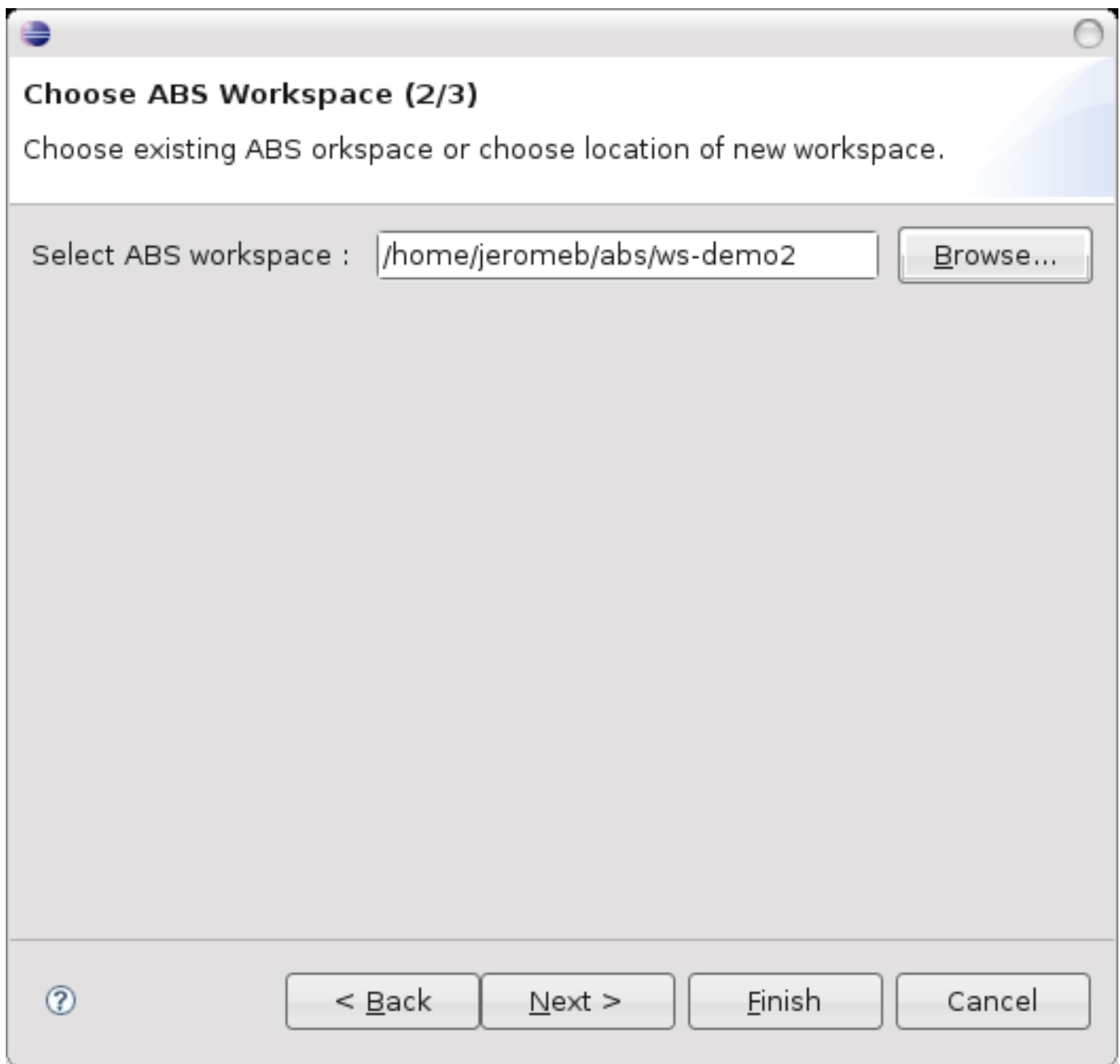


Figure 4.3. Screenshot for ABS workspace selection

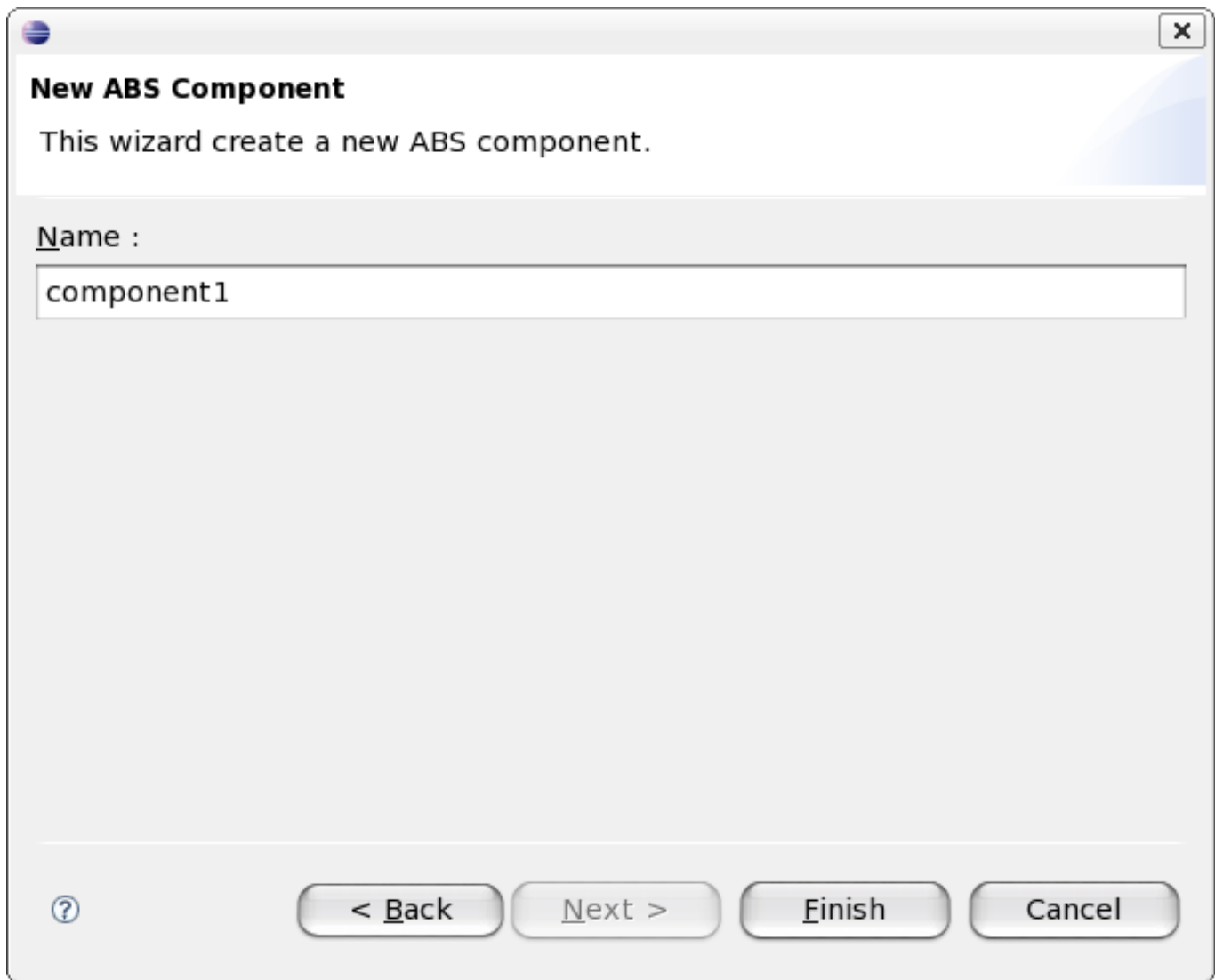


Figure 4.4. Screenshot for component definition

4.2. Creating a new project

In menu : "New" -> "Project" -> in ABS node select "New Project"

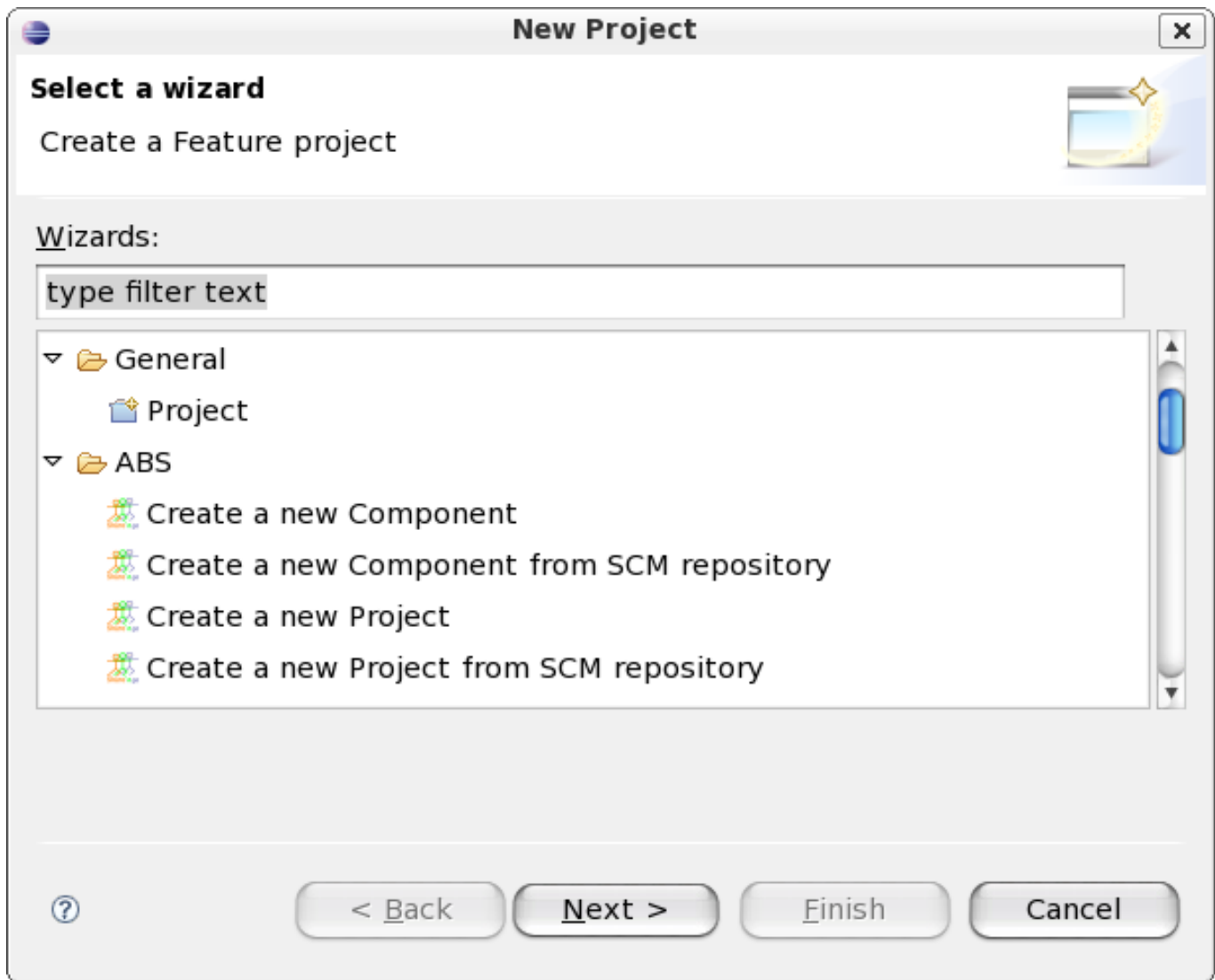


Figure 4.5. Screenshot for project creation

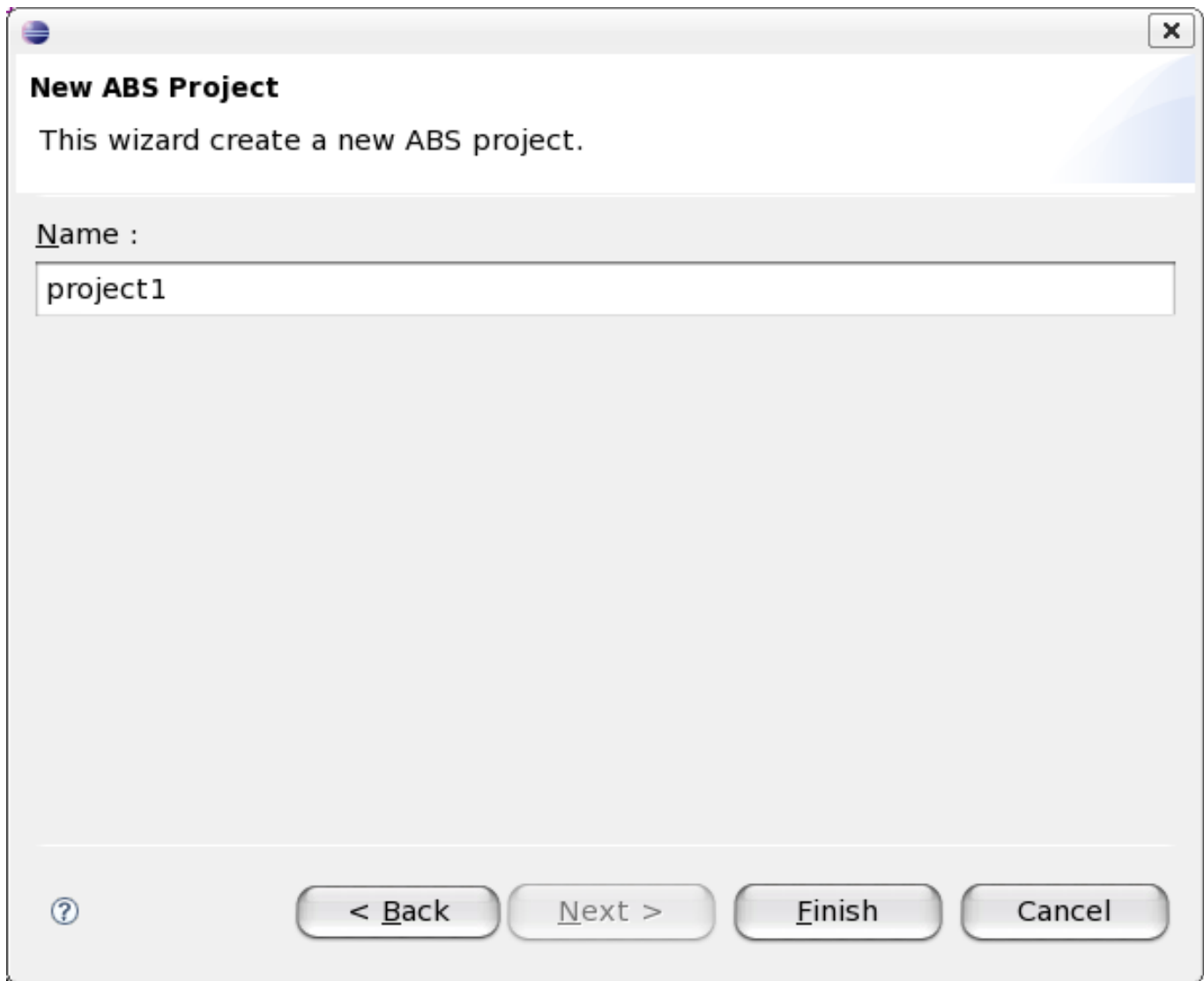


Figure 4.6. Screenshot for project description

When a new project is created, the wizard automatically add a module named "main" and a "localhost" deployment. You'll find in "PackageExplorer" view a module "project1_main" and a deployment "project1_localhost_server".

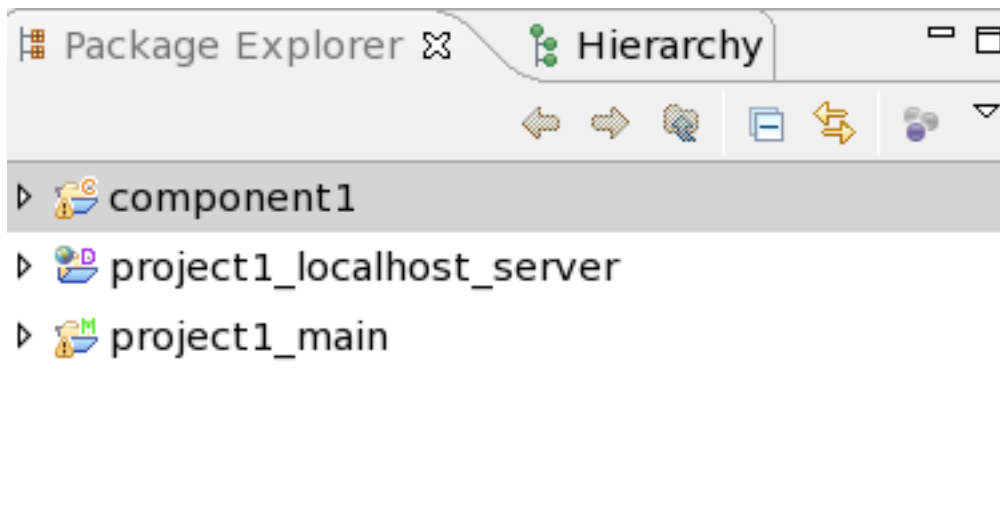


Figure 4.7. Screenshot of package explorer view used to visualize project resources

4.3. Creating a design model

A design model may be created for a component or project module.

4.3.1. UML modelization

Locate yourself in server/model/conception directory and create a model named conception.uml using menu "New" -> "Other..." -> "SharenGo Unified Modeling" -> "Step 3 - New Conception Model"

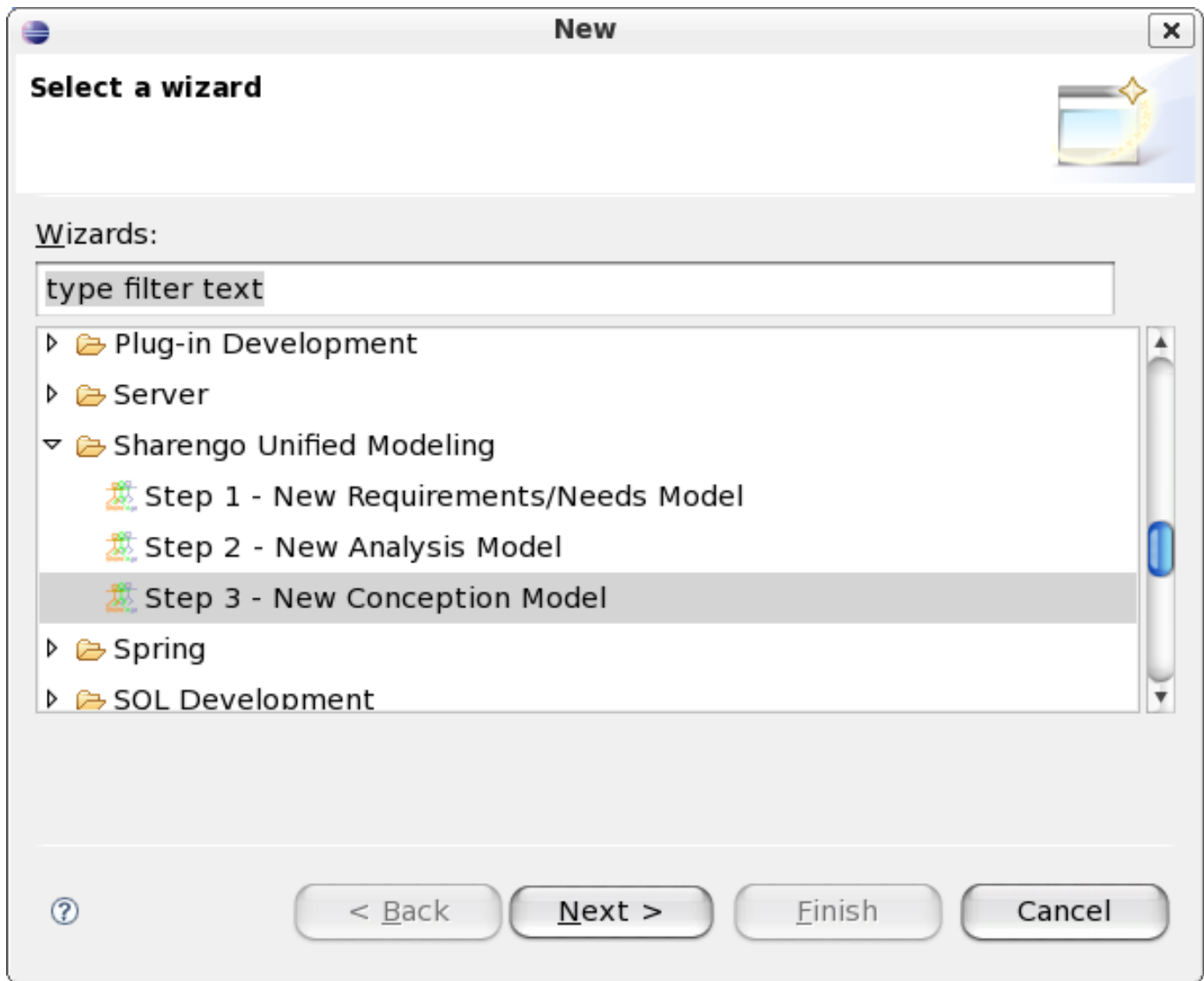


Figure 4.8. Screenshot for design model creation

This action create three files :

- `conception.uml` : this file contains all structural model element,
- `conception.umldi` : this file contains all diagrams description using the first one to reference element appearing in diagrams,
- `conception.properties` : this file is used to parametrize generation strategy. The default values are sensible for current use.

Note

You MUST name the model `org::sharengo::component_name` using the properties windows. (":" is the package separator character used in UML) This notation permits to find the package name containing the generated files.

4.3.2. Logical architecture

In order to modelize the various components we need, we have define a logical architecture at the PIM (Platform Independant Model) level in the MDA way. The following schema shows the different layers of this architecture based on the MVC pattern.

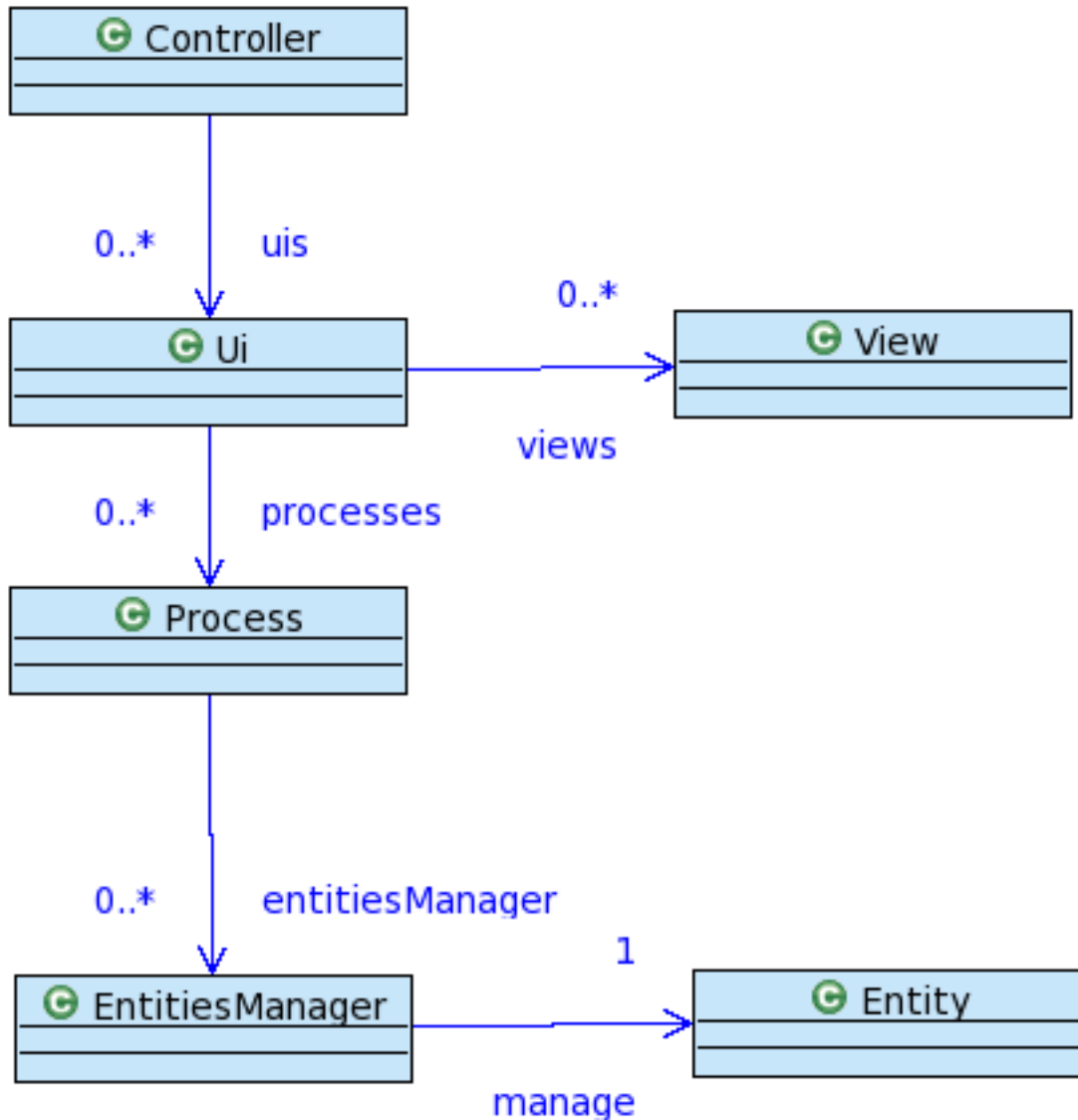


Figure 4.9. Logical architecture representation

This architecture model may be seen as a metamodel because it will be instantiated in each component and module. For historical reasons, we have chosen the UML language and not a DSL (Domain Specific Language). A UML profile has been designed to incorporate the various concepts used in the architecture metamodel. This profile contains a set of stereotypes and properties described in the next section.

4.3.2.1. Stereotypes

Stereotypes are used to add specific semantic meaning to UML elements. The next figure shows how to apply a ste-

reotype in Topcased modeler.

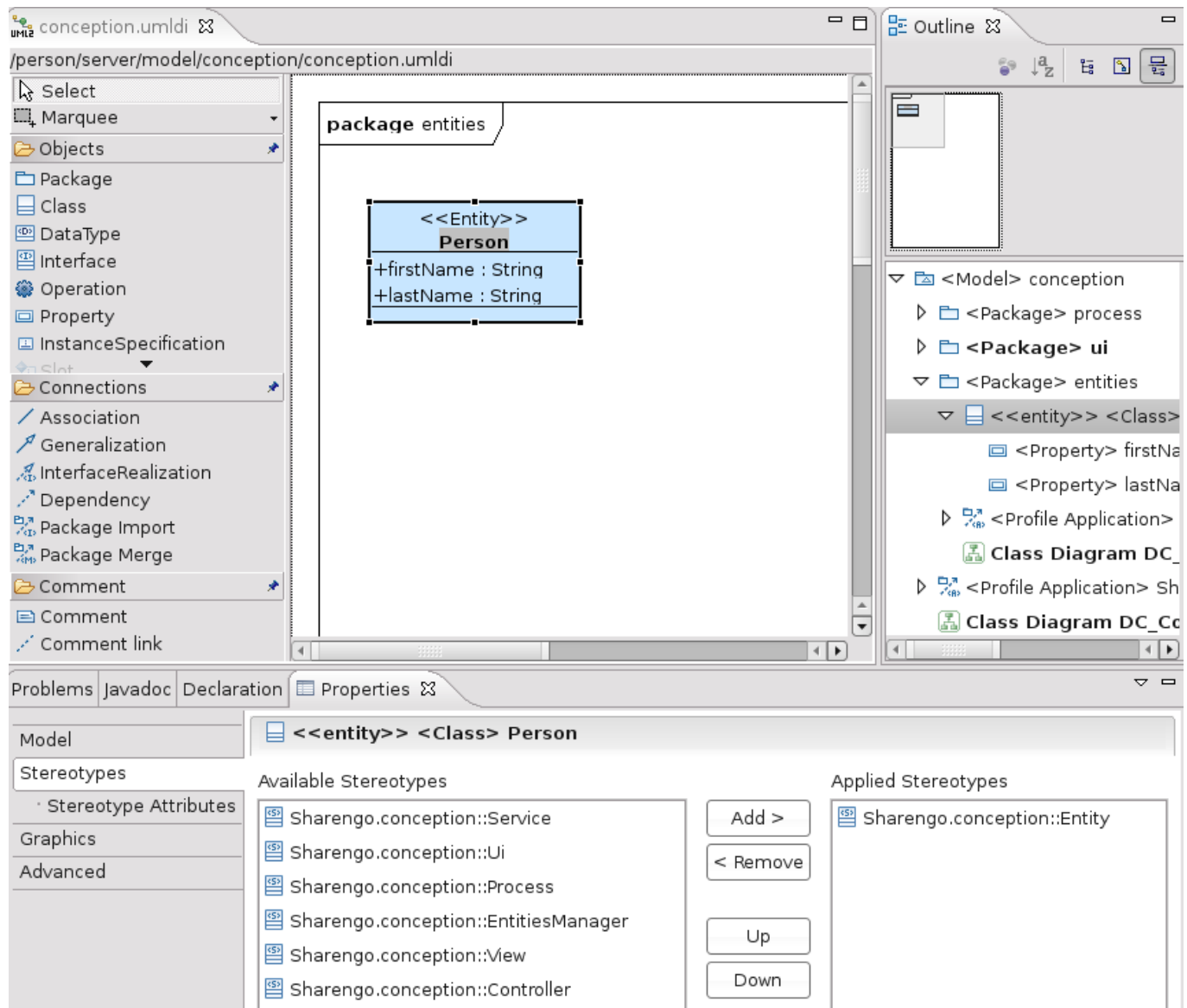


Figure 4.10. Applying a stereotype

A stereotype define a set of properties which can be valued in the "Stereotype attributes" tab.

4.3.2.2. <<Entity>> stereotype

Definition. This stereotype is used to define business domain objects (which are persisted by the application).

Properties. None

Usage context.

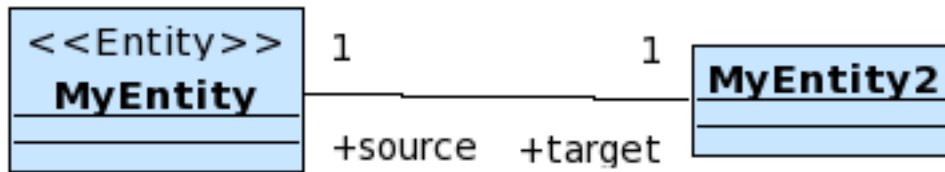


Figure 4.11. Association between 2 entities

4.3.2.3. <<Dto>> stereotype

Definition. This stereotype is used to specify transferable datas. The DTO (Data Transfert Objects) pattern is used to build a mediation layer between business processes and UI layer.

Properties. None

Usage context. Often used as an operation parameter.

4.3.2.4. <<EntitiesManager>> stereotype

Definition. This stereotype is applied on class defining Data Access Objects following the well known pattern. This kind of class manage storage and retrieval for entities.

Properties. None

Dependencies. TODO ajouter un screenshot

4.3.2.5. <<Process>> stereotype

Definition. Used on classes defining business services ou processes

Properties. None

Usage context. A "process" class may used others processes or EntitiesManager

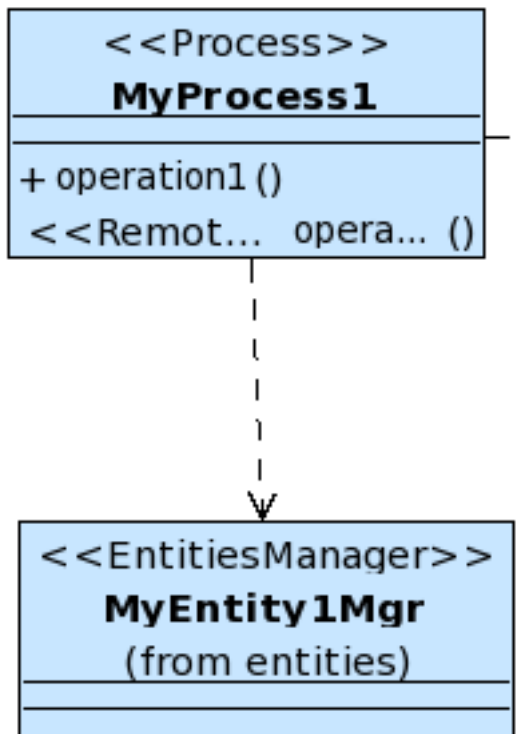


Figure 4.12. Use of EntitiesMgr by a process class

4.3.2.6. <<Controller>> stereotype

Definition. This stereotype is used to route user requests. It decodes transmitted datas and send them to the UI layer.

Properties. None

Usage context.

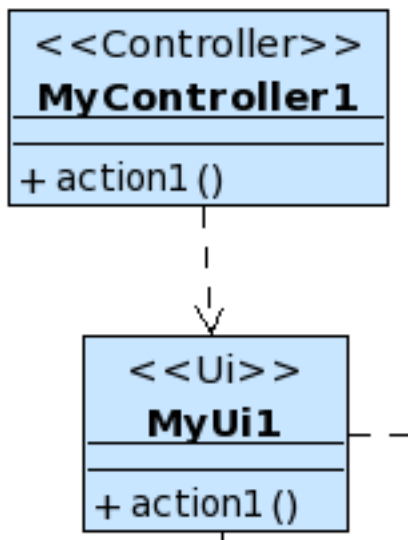


Figure 4.13. Use of UI class by a controller

4.3.2.7. <<Ui>> stereotype

Definition. This stereotype is applied on classes orchestrating requests to business process and transmitting the results to view in order to show datas to end users.

Properties. None

Usage context. A UI class may used others UI, UI uses Process and views Une Ui peu utiliser d'autres Ui, faire appels à des Process et utiliser des View.

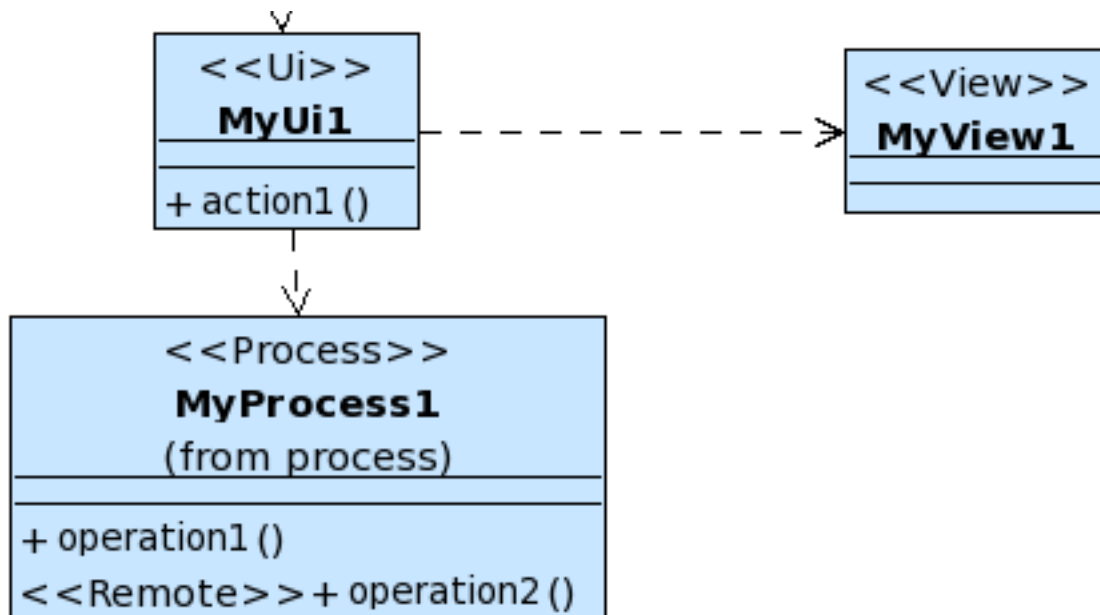


Figure 4.14. Process and view used by UI

4.3.2.8. <<View>> stereotype

Definition. This stereotype is used to define user "screen".

Properties. None

Usage context. A view is used by UI.

4.3.2.9. <<Remote>> stereotype

Definition. This stereotype express that an operation is remotely usable. In a UI context, that means that an AJAX kind of use is possible (call from web browser). In a Process context the operation is published as web service.

Properties. TODO

4.3.2.10. <<Transactional>> stereotype

Definition. Express the transactionnal semantic of a process operation.

Properties. None

4.3.2.11. <<Config>> stereotype

Definition. This stereotype is applied on classes defining configuration values. For example, in order to define a smtp server name, you have to define a "smtpserver" attribute of type String in a class supporting the "Config" stereotype.

Properties. None

Usage context. Config classes may be used by all classes of the architecture.

4.4. Generating code

We are using the Acceleo transformation model to text tool to generate source code from UML model following the logical architecture. In order to run a transformation, you to define a transformation chain. In our case, we will define a specific chain calling a generic one for a given architecture.

To create the chain "conception.launch", select menu "New" -> "Others" -> "Acceleo" -> "Module Launcher" :

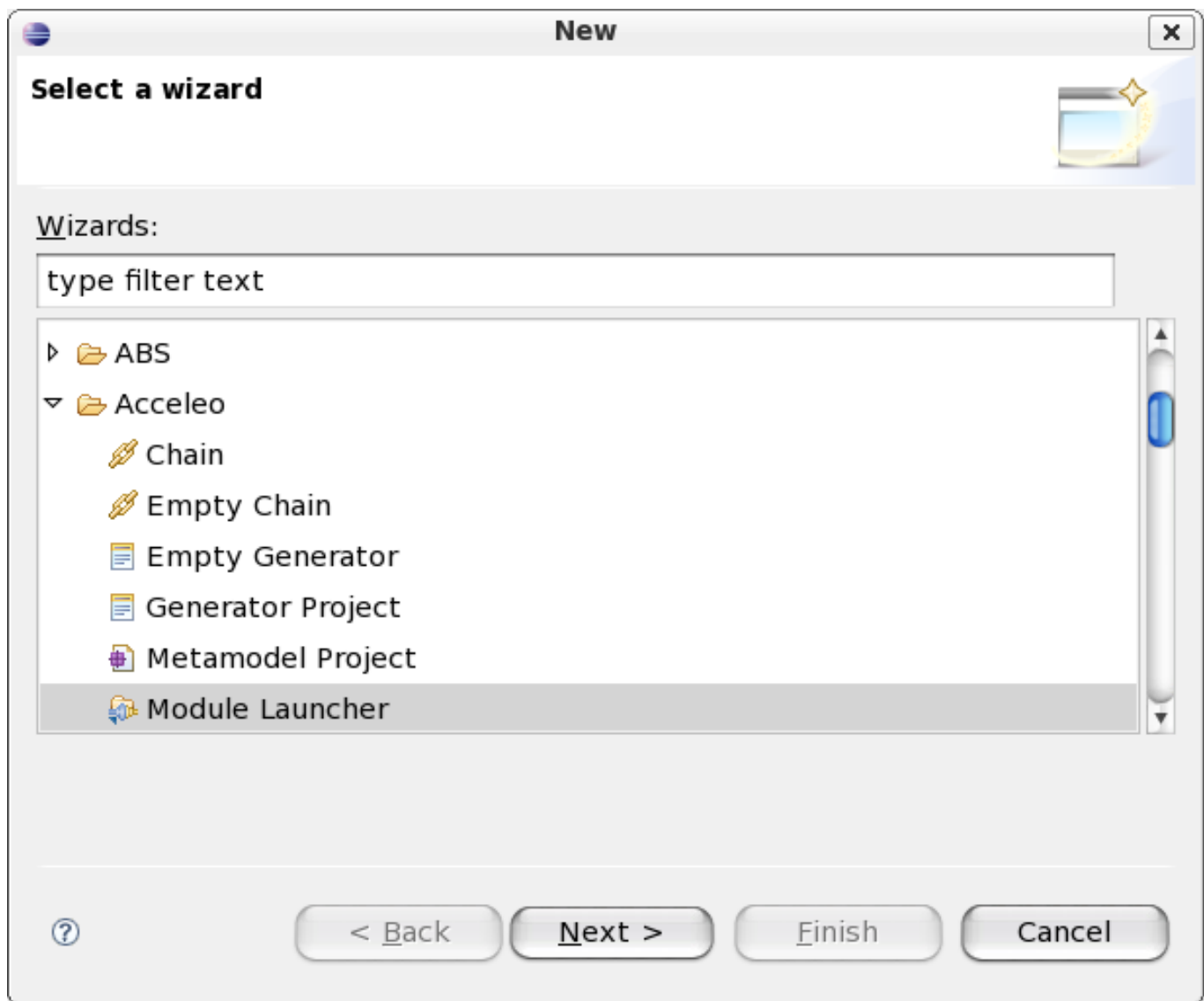


Figure 4.15. Screenshot for etape 1 of ModuleLauncher wizard

Choose chain : "Java Generator for Spring / Hibernate / Velocity Architecture" :

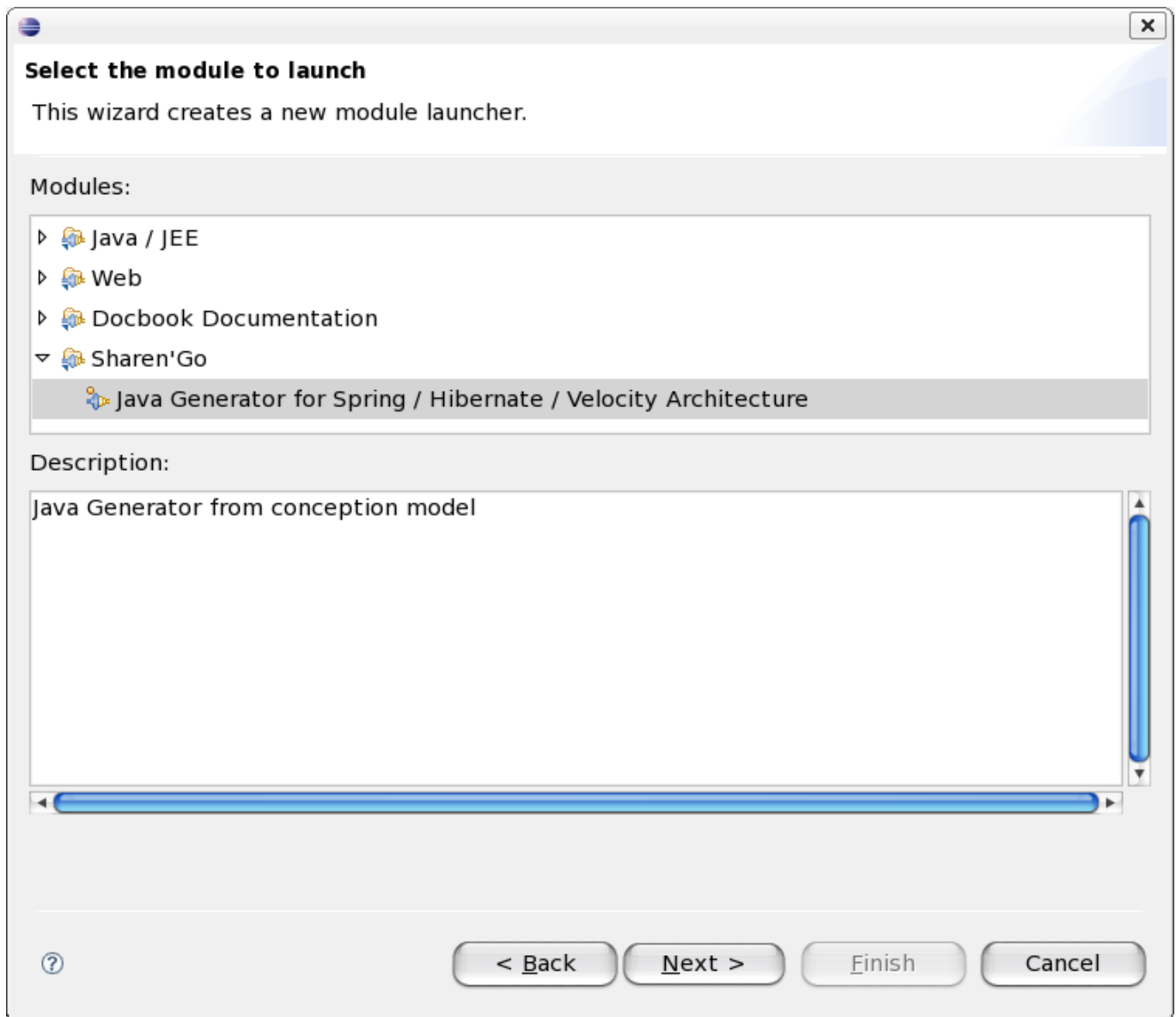


Figure 4.16. Screenshot for etape 2 of ModuleLauncher wizard

Click "Next" and set "conception.chain" as name :

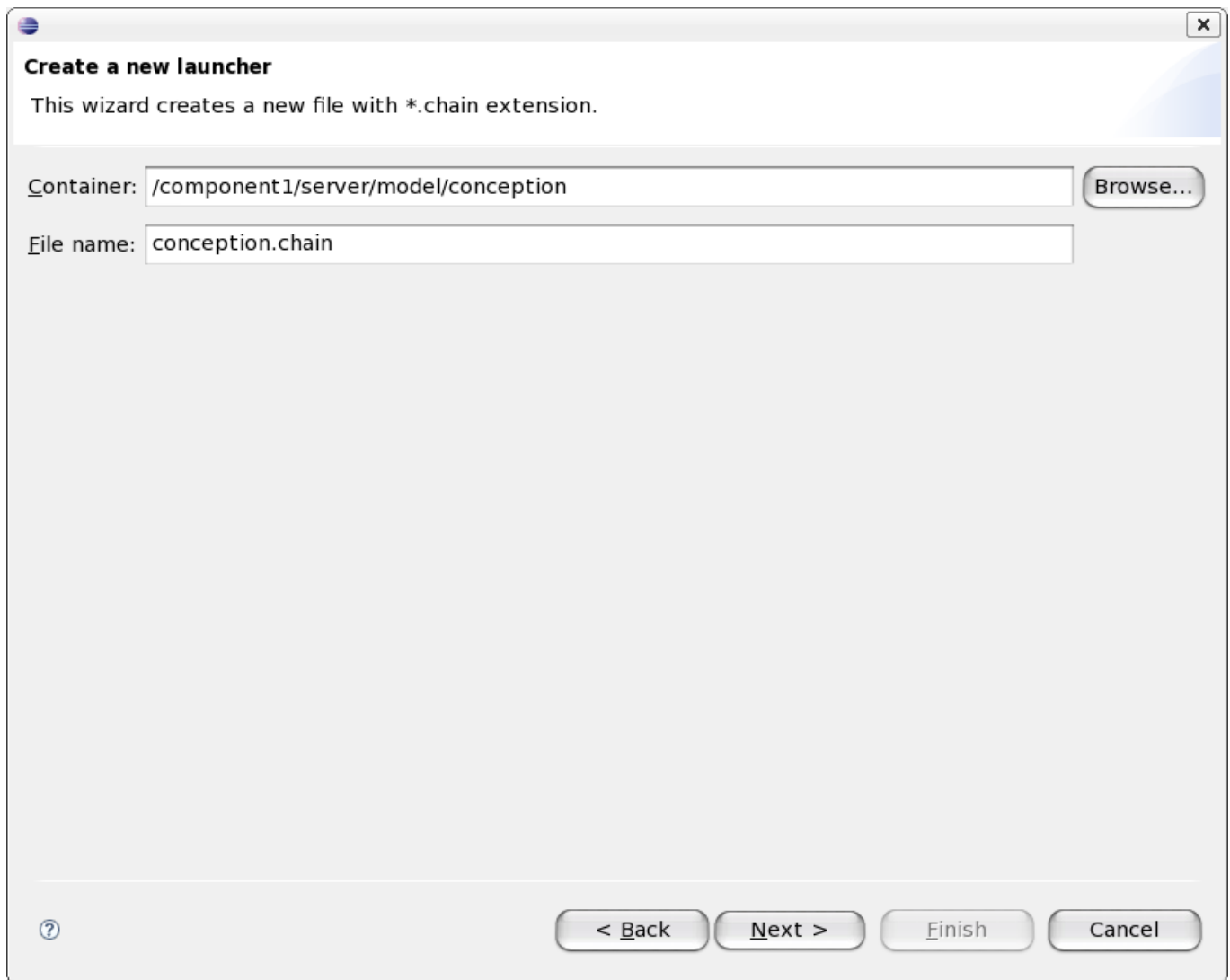


Figure 4.17. Screenshot for etape 3 of ModuleLauncher wizard

click "Next", select the surce UML model source (conception.uml).

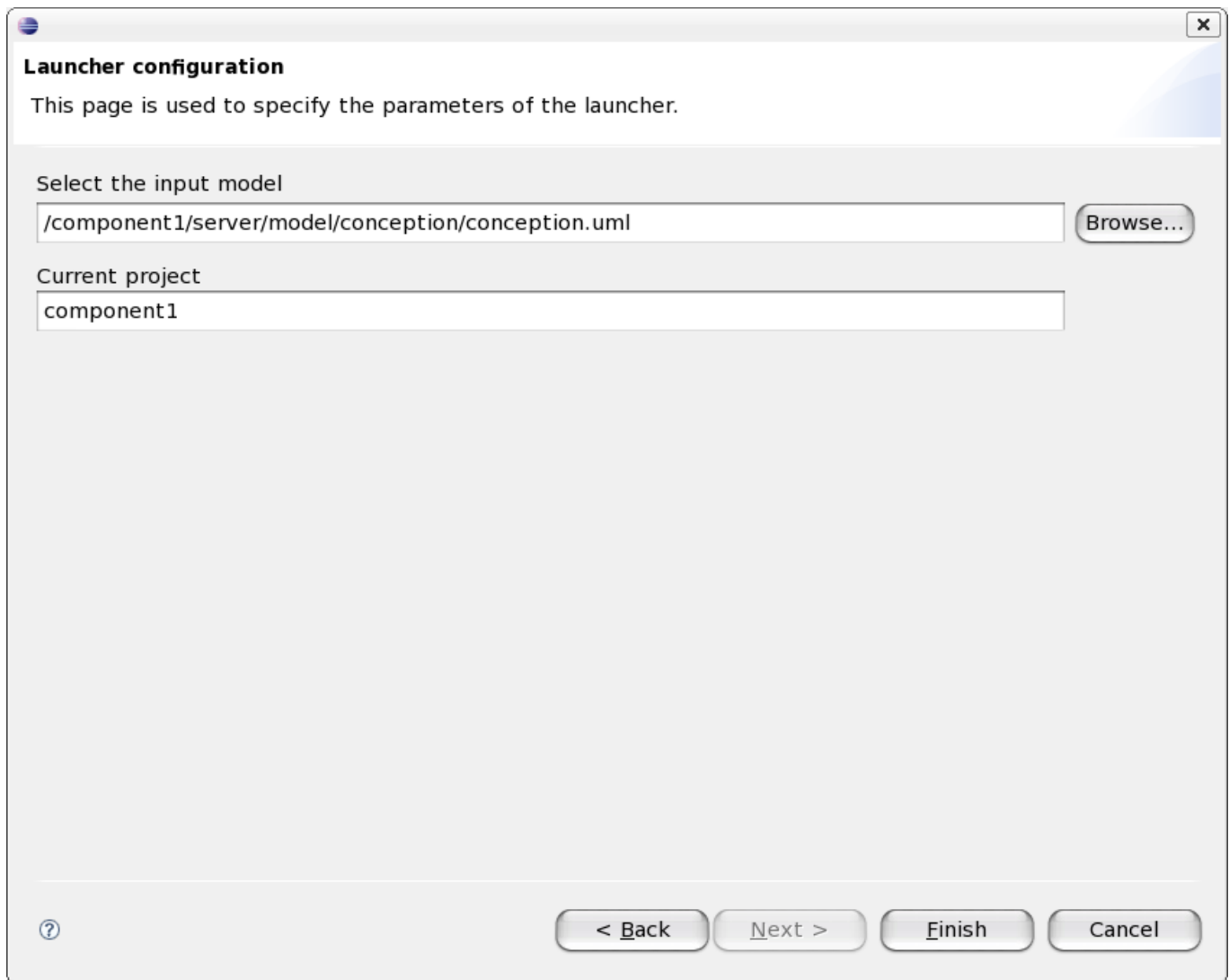


Figure 4.18. Screenshot for etape 4 of ModuleLauncher wizard

Cliquer sur "Next", sélectionner le modèle UML source `conception.uml` :

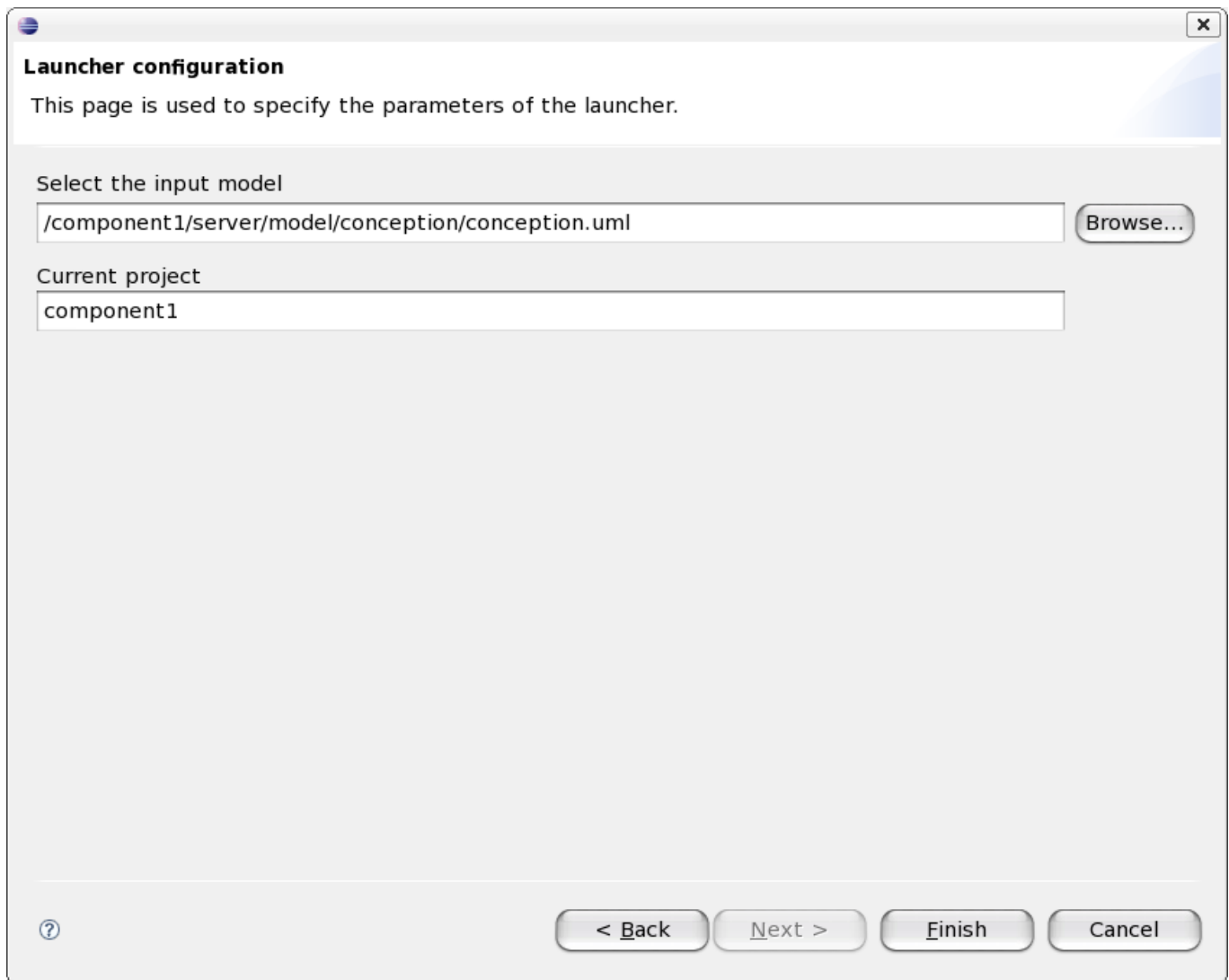


Figure 4.19. Screenshot for etape 5 of ModuleLauncher wizard

Double click on the file to edit the chain if you want to modify something

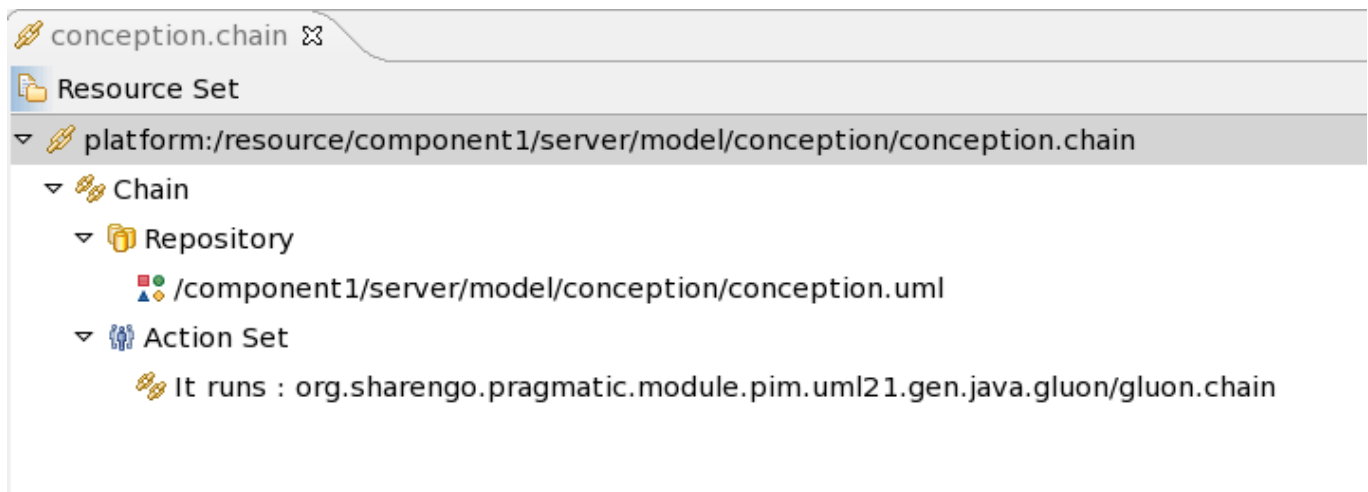


Figure 4.20. Screenshot for chain editing

Run the code source generation choosing the "Launch" option in the contextual menu of the `conception.chain` file.

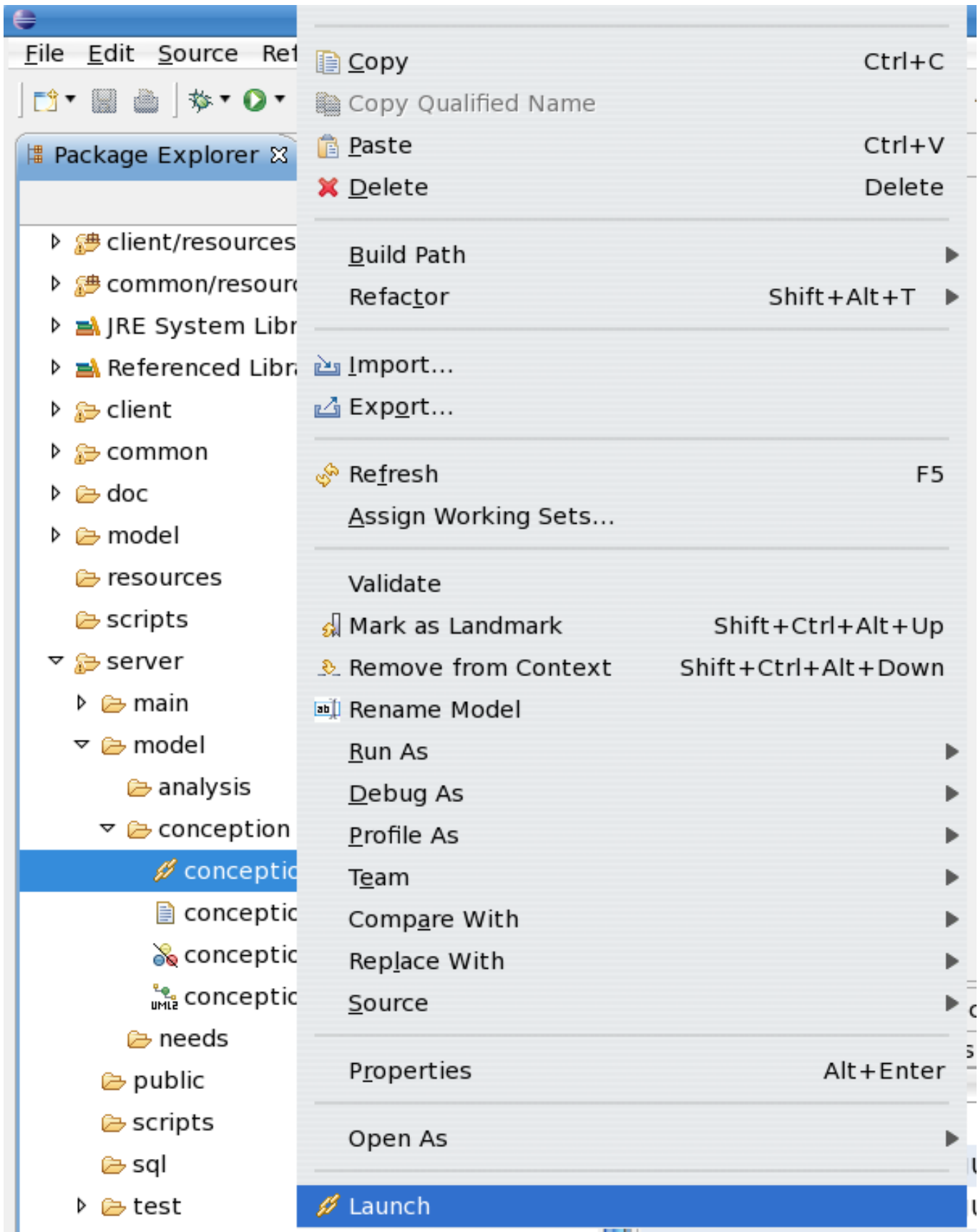


Figure 4.21. Screenshot of generation run

4.4.1. Spring based physical architecture

Spring is a java library used to build technical frameworks for various architecture. It's a light container offering an environment comparable to Spring biggest strength is his use of IoC (Inversion of Control) pattern described here :<http://www.martinfowler.com/articles/injection.html> by famous architect Martin Fowler. The reference documentation of the project is located here : <http://www.springframework.org/docs/reference/>

4.4.1.1. Prerequisite

In order to use Spring in a component or module, you must add a dependency to `middleware/gluon-core` component. This component defines all binary dependencies needed by Spring.

First, you have to retrieve this component from the repository using the option menu `New / Project ... / ABS / Create a new Component from SCM repository` :

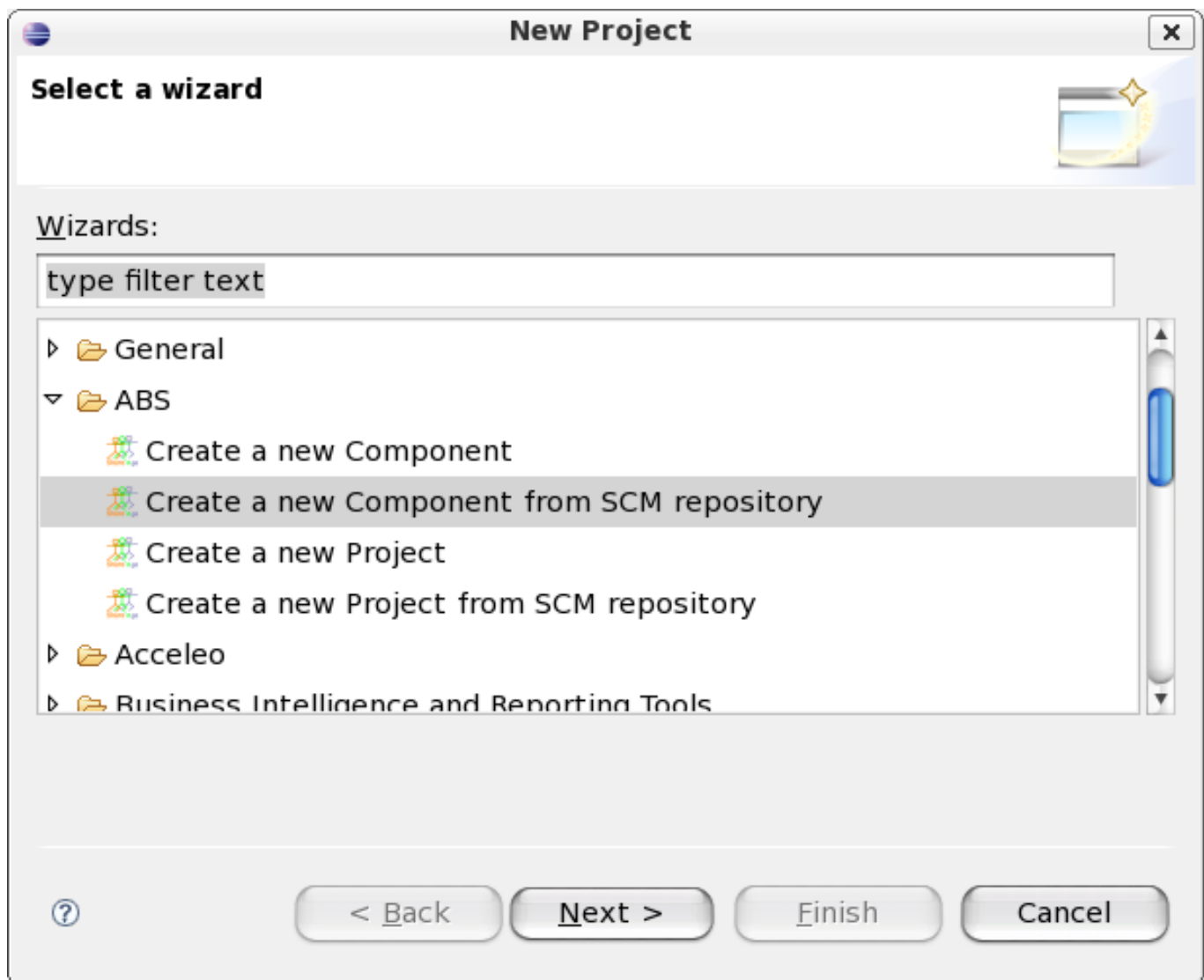
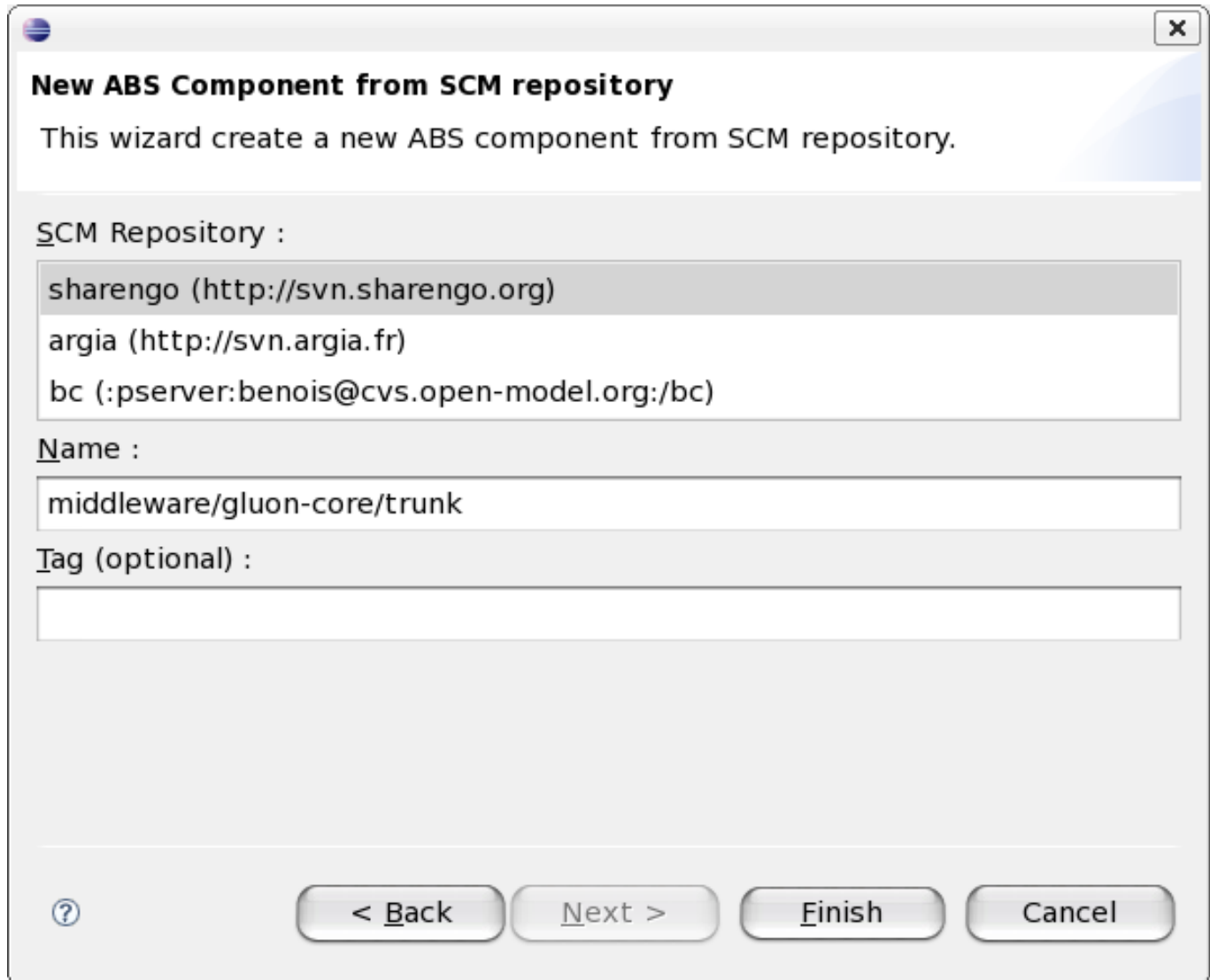


Figure 4.22. Screenshot for retrieving an existing component step 1

Choose the target repository and input the component name. In our case we'll use the trunk version.

**Figure 4.23. Screenshot for retrieving an existing component step 2**

Add the dependency toward this component, editing the `srcdep.txt` file in component or module root directory and adding the following line :

Example 4.1. srcdep.txt file example

```
sharengo:middleware/gluon-core/trunk
```

Note

If you want to use work with web services, you must add one more dependency to the `middleware/gluon-xfire` component.

4.4.1.2. Generated files

You'll find thereafter an exhaustive list of all generated files for each kind of stereotype.

4.4.1.2.1. From <<Entity>>stereotype

For an entity class named "MyEntity1" located in the package "entities" in a "org::sharengo::component1" model , the generated files are :

- server/main/src/org/sharengo/component1/entities/MyEntity1.java : business class
- server/resources/org/sharengo/component1/entities/hibernate/MyEntity.hbm.xml : hibernate mapping file
- server/test/src/org/sharengo/component1/entities/impl/MyEntityHelper.java : helper class used to create business object (used in unit tests)

4.4.1.2.2. From <<Dto>> stereotype

For a dto class named "MyDto1" located in the "dto" package in a "org::sharengo::component1" model, there is only one generated file :

- /server/main/src/org/sharengo/component1/dto/MyDto1.java

4.4.1.2.3. From <<EntitiesManager>> stereotype

For an EntityManager class named "MyEntity1Mgr" located in the package "entities" in a "org::sharengo::component1" model , the generated files are :

- server/main/src/org/sharengo/component1/entities/IMyEntity1Mgr.java : Data Access interface
- server/main/src/org/sharengo/component1/entities/impl/MyEntity1MgrImpl.java : Date Access implementation related to the previous interface
- server/test/src/org/sharengo/component1/entities/impl/MyEntity1MgrTest.java : unit tests for the implementation
- server/resources/META-INF/spring/org.sharengo.component1/layer-daos-hibernate.xml : Spring configuration file with entities manager declaration and related dependencies.

4.4.1.2.4. From <<Process>> stereotype

For a process class named "MyProcess1" located in the "process" package of a "org::sharengo::component1" model, the generated files are :

- server/main/src/org/sharengo/component1/process/IMyProcess1.java : business service interface
- server/main/src/org/sharengo/component1/process/impl/MyProcess1Impl.java : business service implementation

- server/test/src/org/sharengo/component1/process/impl/MyProcess1Test.java : unit tests for implementation
- server/resources/META-INF/spring/org.sharengo.component1/layer-services.xml : Spring configuration file with process class declaration and related dependencies. It also defines the transactional strategy for each operation supported the "Transactional" stereotype.

If some operations support the "Remote" stereotype, more files are generated :

- server/main/src/org/sharengo/component1/process/IMyProcess1WebService.java : interface containing only "remote" methods declaration.
- server/resources/META-INF/spring/org.sharengo.component1/layer-xfire-services.xml : Spring configuration file used to link the interface to the implementation and defining the URL access for each operations.
- server/test/src/org/sharengo/component1/process/impl/MyProcess1WebServiceTest : unit tests for remote methods. Those tests reuses the process unit tests providing a process instance used through remote access using an embedded jetty server.
- server/test/src/org/sharengo/component1/process/impl/MockMyProcess1WebService.java : mock object for unit tests
- client/main/src/org/sharengo/component1/ws/WSClietFactory.java : factory creating IMyProcess1WebService instance from URL like http://localhost/project1/services/IMyProcess1WebService

4.4.1.2.5. From <<Controller>> stereotype

For a controller class named "MyController1" located in a "ui" package in a "org::sharengo::component1", the generated files are :

- server/main/src/org/sharengo/component1/ui/MyController1.java : router class to "UI"
- server/resources/META-INF/spring/org.sharengo.component1/layer-controllers.xml : Spring configuration file containing controller implementation declaration and related dependencies.

4.4.1.2.6. From <<Ui>> stereotype

For a ui class named "MyUi1" located in the "ui" package of a "org::sharengo::component1" model, the generated files are :

- server/main/src/org/sharengo/component1/ui/IMyUi1.java : UI interface
- server/main/src/org/sharengo/component1/ui/impl/MyUi1Impl.java : ui implementation
- server/resources/META-INF/spring/org.sharengo.component1/layer-uis.xml : Spring configuration file containing ui implementation declaration and related dependencies.

4.4.1.2.7. From <<View>> stereotype

For a view class named "MyView1" located in the "ui" package of a "org::sharengo::component1" model, the generated files are :

- `server/main/src/org/sharengo/component1/ui/impl/MyView1.java` : view class
- `server/main/src/org/sharengo/component1/ui/ViewFactory.java` : view object factory. the view objects are stateful and cannot be injected as the others elements of the architecture.

4.4.1.2.8. From <<Config>> stereotype

For a Config class named "MyConfig1" located in the "config" package of a "org::sharengo::component1", the generated files are :

- `server/main/src/org/sharengo/component1/Conf/MyConfig1.java` : configuration class
- `server/resources/META-INF/spring/org.sharengo.component1/layer-configs.xml` : Spring configuration file declaring the previous class.

4.4.1.2.9. From the model itself

- `server/resources/META-INF/spring/component.xml` : Spring configuration files defining what are the needed "layer*" configuration files for the current component
- `server/resources/META-INF/spring/component-test.xml` : same role as previous file but used in unit tests context and therefore modifiable
- `server/resources/META-INF/spring/org.sharengo.component1/applicationContext-tests.xml` : Spring configuration file defining hibernate resources and others resources used in unit tests.
- `server/test/src/org/sharengo/component1/AbstractBusinessLayerTests.java` : abstract class used by all unit tests. It's used to initialize the components graphs.

4.5. Testing generated code

4.5.1. technical layers configuration

Before running unit tests, we have to verify and adjust component parametrization in `server` root directory / `resources/META-INF/spring` :

4.5.1.1. component.xml file

This file is the entry point of all components and modules. It defines all the prerequisites for optimal execution of each components. It defines all technical and business layers which must be loaded at startup time. You have to modify this file in relation with usage (component with or without ui, with or without WebServices facade, etc).

Example 4.2. component.xml example



```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/s

<!-- Start of user code component definition -->

<!-- load component Controller layer -->
<import resource="classpath:META-INF/spring/org.sharengo.component1/layer-controllers.xml"/>
<!-- load component Ui layer -->
<import resource="classpath:META-INF/spring/org.sharengo.component1/layer-uis.xml"/>
<!-- load component Process layer -->
<import resource="classpath:META-INF/spring/org.sharengo.component1/layer-services.xml"/>
<!-- load component EntityManager layer -->
<import resource="classpath:META-INF/spring/org.sharengo.component1/layer-daos-hibernate.xml"/>
<!-- load component Config layer -->
<import resource="classpath:META-INF/spring/org.sharengo.component1/layer-configs.xml"/>
<!-- load component Remote / Web Services layer -->
<import resource="classpath:META-INF/spring/org.sharengo.component1/layer-xfire-services.xml"/>

<!-- technical layers -->
<!-- load Hibernate layer -->
<import resource="classpath:META-INF/spring/layer-hibernate.xml"/>
<!-- load Velocity layer -->
<import resource="classpath:META-INF/spring/layer-velocity.xml"/>
<!-- load XFire layer -->
<import resource="classpath:META-INF/spring/layer-xfire.xml"/>

<!-- End of user code component definition -->

</beans>

```

4.5.1.2. component-test.xml file

This file load all needed components and resources needed for unit testing the current component. The default configuration is sensible and must be sufficient, but you can modify it.

Example 4.3. component-test.xml example

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/s

<!-- Start of user code component definition for test environment -->

<!-- include all component.xml file found in classpath to startup all needed components -->
<import resource="classpath*:META-INF/spring/component.xml"/>

<!-- load needed resources for unit tests -->
<import resource="classpath:META-INF/spring/org.sharengo.component1/applicationContext-tests.xml"/>

<!-- End of user code component definition for test environment -->

</beans>

```

4.5.1.3. applicationContext-tests.xml file

It's located in directory `server/resources/META-INF/spring/MODEL_NAME/` and specify databases resources among others if needed. You may adjust the "Start of user code other hbm files" section in order to load mapping files from others components.

Example 4.4. applicationContext-tests.xml example

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schem
                          http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
  <property name="url" value="jdbc:hsqldb:."/>
  <property name="username" value="sa"/>
  <property name="password" value=""/>
</bean>

<!-- ===== HIBERNATE CONFIGURATION ===== -->

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="dataSource" ref="dataSource"/>
  <property name="mappingResources">
    <list>
      <value>org/sharengo/component1/entities/hibernate/MyEntity.hbm.xml</value>
      <!-- Start of user code other hbm files-->
      <!-- End of user code -->
    </list>
  </property>
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect">org.hibernate.dialect.HSQLDialect</prop>
      <prop key="hibernate.show_sql">true</prop>
      <prop key="hibernate.generate_statistics">true</prop>
      <prop key="hibernate.hbm2ddl.auto">create</prop>
      <prop key="hibernate.jdbc.batch_size">1</prop>
    </props>
  </property>
</bean>

<!-- Start of user code specific test injection-->

<!-- End of user code specific test injection -->

</beans>
```

4.5.2. Testing data access layer

The `server/test/src/org/sharengo/component1/entities/impl/MyEntity1MgrTest.java` class implements a unit tests to check basics (CRUD) data access operations. The class must be modify to include new tests each time a new operation is added in the EntitiesManager class. Then you can run the unit test :

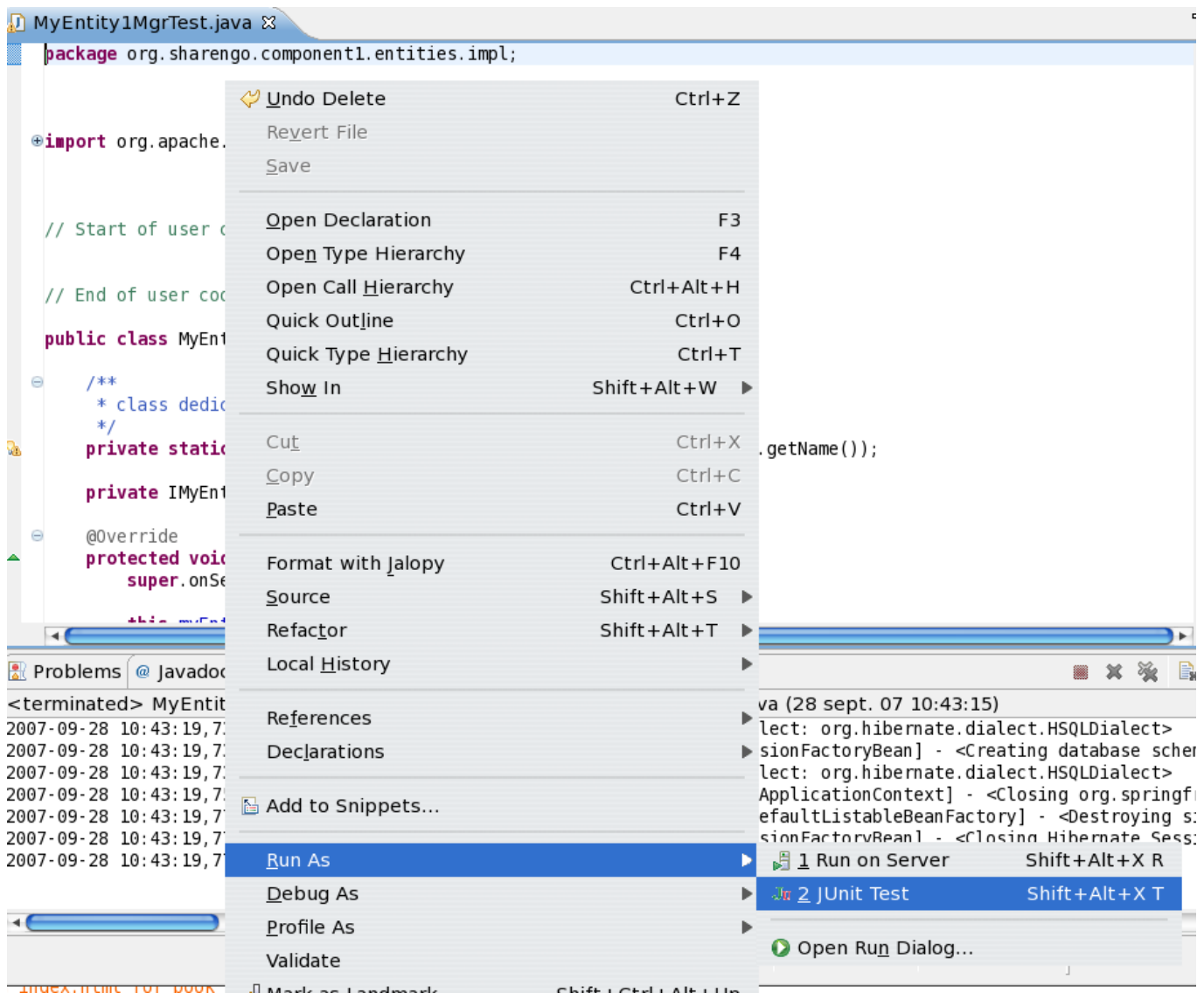


Figure 4.24. Screenshot for unit tests run

and check results :

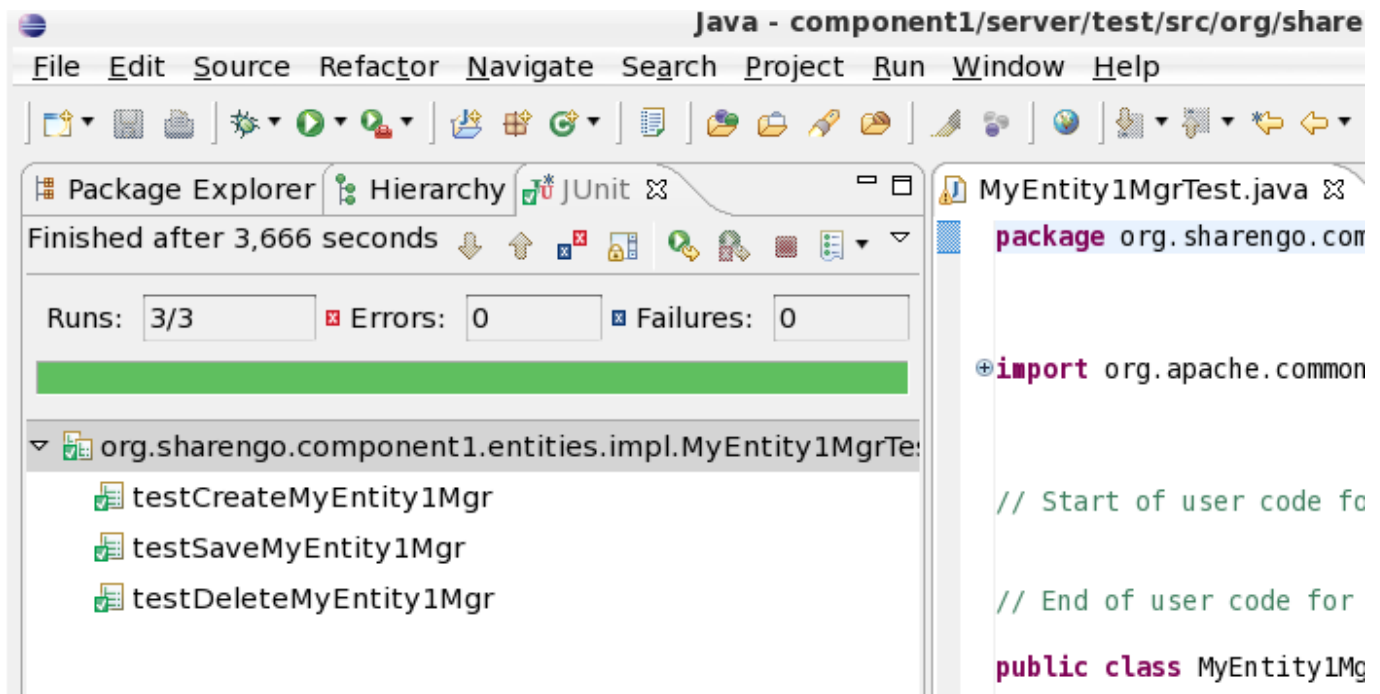


Figure 4.25. Screenshot for unit tests results

4.5.3. Testing business process layer

The class `server/test/src/org/sharengo/component1/process/impl/MyProcess1Test.java` implements unit tests to check business process layer operation. You have to code a test, the tested operations and run unit test.

4.5.4. Testing Web Services

The class `server/test/src/org/sharengo/component1/process/impl/MyProcess1WebServiceTest.java` implement unit tests to check business process layer operation in web service mode.. Run the tests as the others to check specific problems related to communication context.

4.6. Running a project inside Tomcat

4.6.1. Configure your project for webapps deployment. déployer en tant qu'application Web

The `/${PROJECT_NAME}` variable must be replace by the project name in the example files.

4.6.1.1. web.xml file

The web.xml file is automatically generated by assembly of the differents components descriptors files. The deployment task "deploy:build.war" is in charge of this operation. The eclipse plugin call this task during each tomcat startup.

4.6.1.2. Database access configuration

You have to define the variable `@@hibernate.datasource@@` in `${PROJECT_NAME}/deployments/localhost/server/replace.server.properties`

```
@@hibernate.datasource@@=jdbc/${PROJECT_NAME}
```

Remember to add the needed JDBC driver in `tomcat/common/lib` directory and edit the `${PROJECT_NAME}/modules/main/server/main/conf/context.xml` file to define parameters values for database access :

```
<Context
  path="/${PROJECT_NAME}"
  debug="1"
  reloadable="true">

  <Resource
    name="@@hibernate.datasource@"
    auth="Container"
    type="javax.sql.DataSource"
    driverClassName="org.postgresql.Driver"
    url="jdbc:postgresql://localhost:5432/${PROJECT_NAME}"
    username="postgres"
    password=""
    maxIdle="2"
    maxActive="4"
    maxWait="5000"
    validationQuery="select now();"

  />
</Context>
```

Create and edit the file `${PROJECT_NAME}/modules/main/server/main/conf/applicationContext.xml` in order to configure hibernate and specifying all mapping files to load :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jee="http://www.springframework.org/schema/jee"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schem
    http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee

  <import resource="classpath*:META-INF/spring/component.xml" />

  <jee:jndi-lookup id="dataSource" jndi-name="java:comp/env/@@hibernate.datasource@" />

  <bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="mappingResources">
      <list>
        <value>org/sharengo/${OneComponent}/entities/hibernate/${OneHibernateMappingFile}.hbm.xml</value>
      </list>
    </property>
    <property name="hibernateProperties">
      <props>
        <prop key="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</prop>
        <prop key="hibernate.show_sql">true</prop>
        <prop key="hibernate.generate_statistics">true</prop>
        <prop key="hibernate.hbm2ddl.auto">update</prop>
        <prop key="hibernate.jdbc.batch_size">1</prop>
      </props>
    </property>
```

```
</bean>  
</beans>
```

Create the related Postgresql database :

```
$> createdb -U postgres -E unicode project1
```

Add needed dependencies to components used by the project in the `srcdep.txt` file of project module.

4.6.2. Run Tomcat inside Eclipse

Run the Tomcat container using the contextual menu "Run As / Run On Server" in deployment :

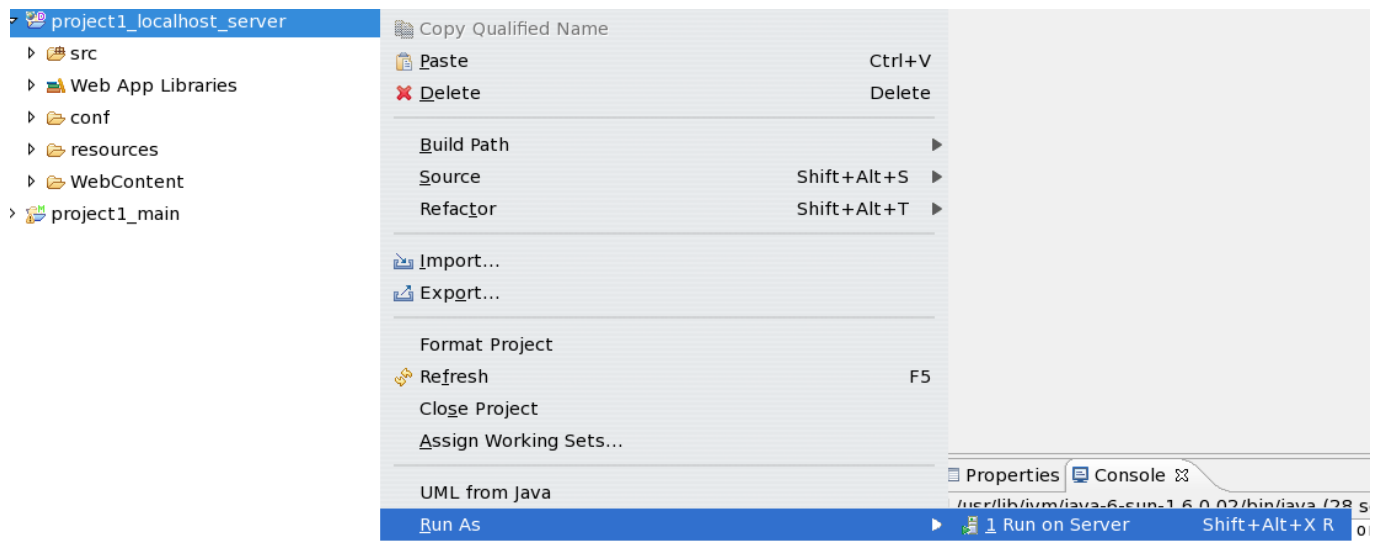


Figure 4.26. Screenshot of running tomcat in eclipse step 1

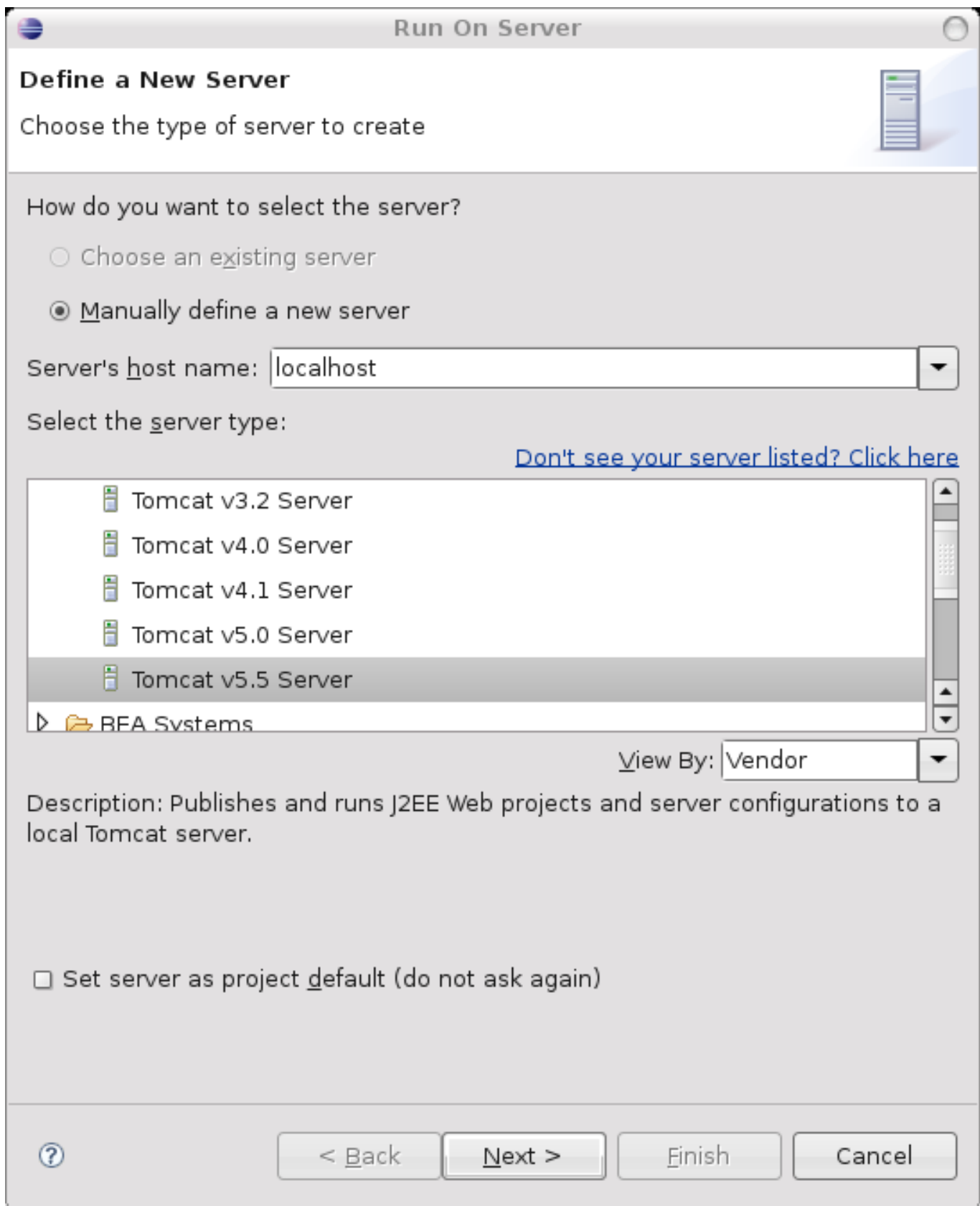


Figure 4.27. Screenshot of running tomcat in eclipse step 2

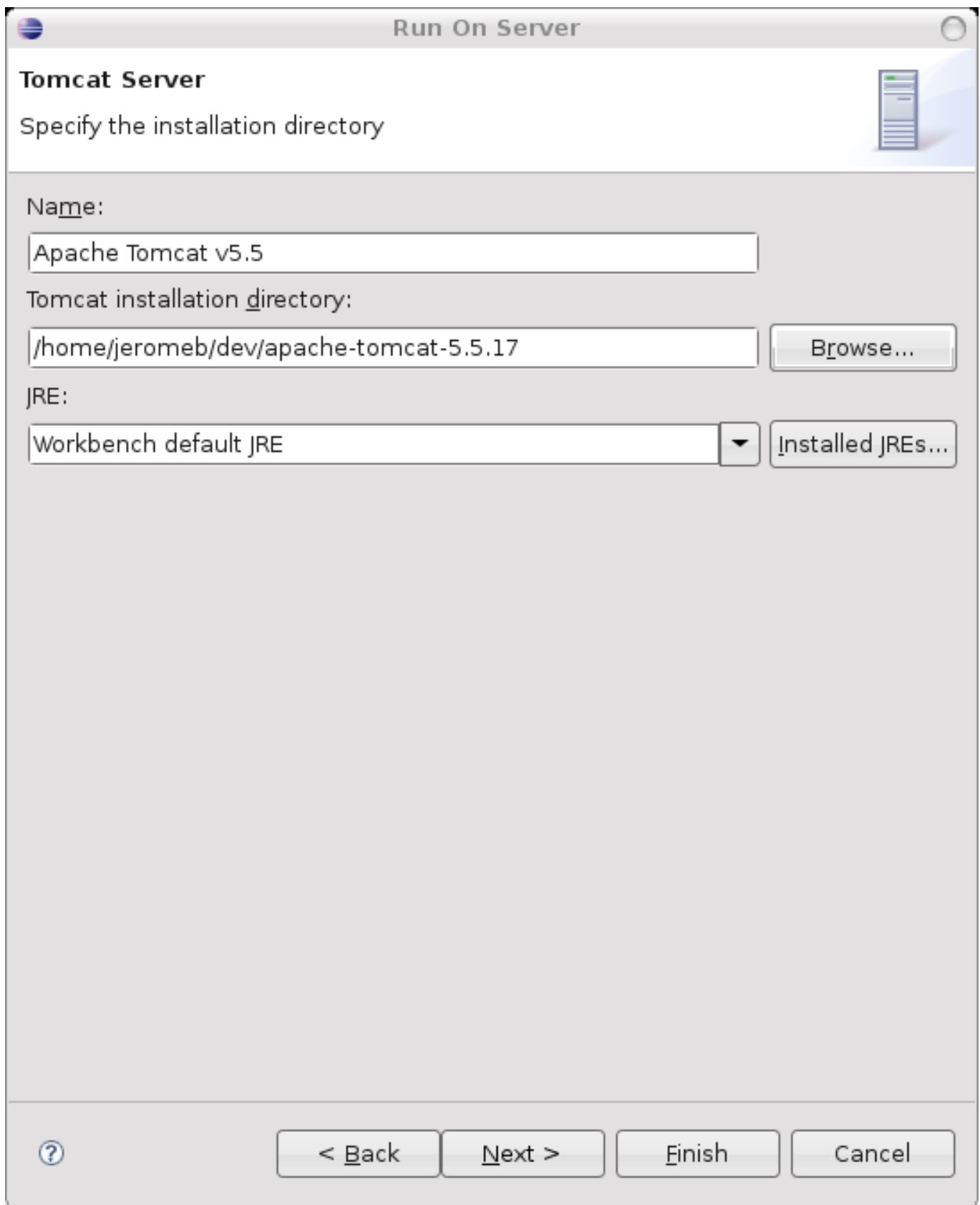


Figure 4.28. Screenshot of running tomcat in eclipse step 3