

Advanced Build System

ABS - Reference Documentation

Release 0.7

Table des matières

Preface	iv
1. Principe de base	1
1.1. Qu'est-ce-qu'un composant ?	1
1.2. Qu'est-ce-qu'une dépendance ?	2
1.2.1. Dépendance binaire	2
1.2.2. Dépendance source	2
1.3. Qu'est-ce-qu'un projet ?	3
1.4. Qu'est-ce-qu'un déploiement ?	3
1.5. Qu'est-ce-qu'un module ?	4
1.6. Qu'est-ce-qu'une tâche/activité ?	4
2. Installation	5
2.1. Installation basique de l'ABS	5
2.2. Installer le Plugin Eclipse ABS	6
3. Utilisation en ligne de commande	9
3.1. Qu'est-ce qu'un workspace	10
3.2. Créer un projet	11
3.2.1. Créer un nouveau projet	11
3.2.2. Récupérer un projet existant	12
3.2.3. Récupérer un projet existant dans une version donnée	12
3.3. Créer un composant	13
3.3.1. Créer un nouveau composant	13
3.3.2. Récupérer un composant existant	13
3.3.3. Récupérer un composant existant dans une version donnée	14
3.4. Compiler	14
3.5. Créer un déploiement	15
3.6. Construire un livrable projet	15
4. Utilisation sous Eclipse	16
4.1. Créer un composant	16
4.2. Création d'un projet	19
4.3. Créer un modèle de conception	22
4.3.1. Modéliser en UML	22
4.3.2. L'architecture logique	24
4.3.2.1. Les stéréotypes	24
4.3.2.2. Le stéréotype <<Entity>>	25
4.3.2.3. Le stéréotype <<Dto>>	26
4.3.2.4. Le stéréotype <<EntitiesManager>>	26
4.3.2.5. Le stéréotype <<Process>>	26
4.3.2.6. Le stéréotype <<Controller>>	27
4.3.2.7. Le stéréotype <<Ui>>	28
4.3.2.8. Le stéréotype <<View>>	28
4.3.2.9. Le stéréotype <<Remote>>	28
4.3.2.10. Le stéréotype <<Transactional>>	29
4.3.2.11. Le stéréotype <<Config>>	29
4.4. Générer du code	29

4.4.1. L'architecture physique basée sur Spring	37
4.4.1.1. Les pré-requis	37
4.4.1.2. Les fichiers générés	38
4.5. Tester le code généré	41
4.5.1. Configurer les couches techniques	41
4.5.1.1. Le fichier component.xml	41
4.5.1.2. Le fichier component-test.xml	42
4.5.1.3. Le fichier applicationContext-tests.xml	43
4.5.2. Tester la couche d'accès aux données	43
4.5.3. Tester les process métiers	45
4.5.4. Tester les Web Services	45
4.6. Lancer le projet dans tomcat	45
4.6.1. Configurer votre projet pour le déployer en tant qu'application Web	45
4.6.1.1. Le fichier web.xml	45
4.6.1.2. Configuration de l'accès à la base de donnée	46
4.6.2. Lancer Tomcat depuis Eclipse	47

Preface

ABS signifie Advanced Build System, il s'agit une chaîne de production orientée composant.

Il ne s'agit pas d'un clône de Maven. Elle se différencie par ses fonctionnalités évoluées d'assemblage de composant. L'ABS se veut simple et pragmatique, son but premier est d'automatiser, de normaliser, d'orchestrer les tâches du cycle de construction d'un applicatif. Il sert à également à mettre en application une méthodologie de conception dirigée par les modèles.

L'ABS intègre différents outils libre du monde java afin de fournir un point d'entrée unique et simple pour chaque activités référencée dans notre méthodologie d'assemblage de projet. L'ensemble des artefacts permettant d'industrialiser chacune de ces activités est exécutable via Ant ou à travers un IDE. Il existe notamment un plug-in Eclipse assez avancé.

L'environnement de développement ABS fournit une manière simple et pragmatique pour construire des projets "MDA" avec une approche composant. Les différentes activités industrialisées à ce jour sont :

- Création de composants ou projets
- Génération de documentation à partir des modèles UML (basée sur PragMatic et Acceleo)
- Gestion des dépendances binaires et inter composants
- Intégration aux outils de gestion de configuration tels que CVS et Subversion
- Compilation des sources
- Lancement des tests unitaires
- Génération de code
- Intégration continue
- Gestion des déploiements grâce à un système de construction de livrables
- Et bien d'autres choses ...

Le projet ABS est un projet libre, collaboratif, vous trouverez plus de détails sur son site web [<http://www.sharengo.org/Wiki?ABS>] : les dernières versions et autres tutoriaux. Vous trouverez également des mailing lists pour poser des questions, partager vos expériences avec d'autres utilisateurs ainsi que les développeurs du projet.

Principe de base

Comme nous l'avons vu en introduction, l'ABS permet d'industrialiser notre méthodologie de réalisation de projet. Pour ce faire nous avons défini un métamodèle, un modèle objet représentant l'ensemble des concepts définissant un projet : projet, composant, dépendance, déploiement, ... Dans cette partie nous nous attacherons à définir chacun de ces concepts.

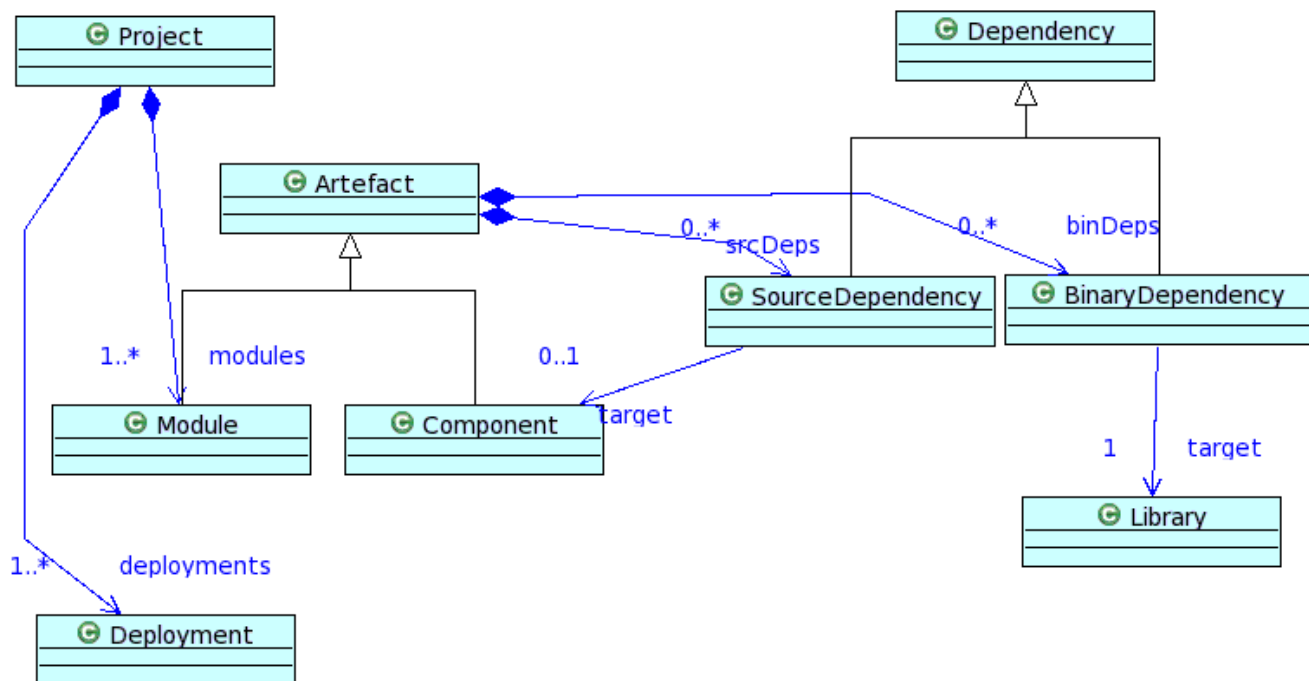


Figure 1.1. Métamodèle ABS

1.1. Qu'est-ce-qu'un composant ?

La programmation orientée composants (POC) consiste à utiliser une approche modulaire au niveau de l'architecture d'un projet informatique, ce qui permet d'assurer au logiciel une meilleure lisibilité et une meilleure maintenance. On privilégie donc l'utilisation de briques réutilisables par opposition à des briques monolithiques. Dans notre contexte, un composant est :

- une entité métier cohérente. Par exemple : Les composants Client, Facture, Commande,
- il est défini par un modèle,

- il expose des services métiers réutilisables dans plusieurs contextes.
- il contient du code source java, des fichiers de configuration générés à partir de son modèle. Cet ensemble de fichiers constitue l'implémentation de référence. Toutefois il est tout à fait envisageable d'utiliser ce même modèle pour générer le code source vers une autre technologie cible.

1.2. Qu'est-ce-qu'une dépendance ?

1.2.1. Dépendance binaire

Un dépendance binaire définit une relation d'utilisation entre un composant A et une librairie tierce. Un module de projet (notion que nous aborderons plus loin) peut également utiliser ce type de dépendance. Les dépendances binaires d'un composant ou module sont stockés dans le fichier `bindep.txt` situé à la racine de l'arborescence.

Exemple 1.1. Un exemple de fichier `bindep.txt`

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

Ces dépendances sont utilisées principalement lors des phases de compilation et pour préparer l'environnement des applications à déployer. Les librairies sont automatiquement téléchargées dans le répertoire locale `ABS_HOME/binary-repository` si nécessaire.

ABS sait où trouver les librairies manquantes et toù les stocker après téléchargement en utilisant des propriétés stockées dans le fichier `build.properties` situé dans le répertoire `ABS_HOME`.

1.2.2. Dépendance source

Une dépendance source définit une relation d'utilisation entre 2 composants : un composant A a besoin d'un composant B pour fonctionner. Un module de projet (notion que nous aborderons plus loin) peut également utiliser ce type de dépendance. Les dépendances sources d'un composant ou module sont stockés dans le fichier `srcdep.txt` situé à la racine de l'arborescence.

Exemple 1.2. Un exemple de fichier `srcdep.txt`

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

L'exemple ci-dessus illustre la déclaration de 2 dépendances. Le format utilisé est `scm_repository:le_nom` :

- `scm_repository` : le nom logique du référentiel CVS ou subversion utilisé. Ce nom est défini dans le fichier `ABS_HOME/repositories.properties` cf. section TODO

- `::` séparateur
- `le_nom` : le nom du composant. La 2ème ligne de cet exemple référence un composant issu d'un référentiel subversion qui fourni implicitement le versionning des dépendances

1.3. Qu'est-ce-qu'un projet ?

Un projet est une application, il constitue le livrable finale. Il résulte de l'assemblage de plusieurs composants métiers. Il définit les environnements cibles appelés déploiements, ainsi que leur spécificités en terme de configuration.

1.4. Qu'est-ce-qu'un déploiement ?

Un déploiement est un environnement cible utilisé pour livrer les war, tar.gz, jnlp. Il est possible de définir plusieurs déploiements au sein d'un même projet. Par exemple : Développement, Test, Pré-production, production ... Les déploiements sont stockés dans le fichier `deployment.txt` situé à la racine de l'arborescence du projet.

Exemple 1.3. Un exemple de fichier `deployment.txt`

```
localhost
test
preprod
prod
```

Le fichier ci-dessous illustre la déclaration de 4 déploiements. Très souvent les environnements présentent des variantes en terme de configuration. Par exemple le serveur SMTP à utiliser pour envoyer un mail peut différer entre 2 environnements. Il est donc nécessaire de variabiliser cette information à l'aide d'un token de remplacement.

Exemple 1.4. Le fichier de propriétés contenant la définition de l'adresse du serveur SMTP

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

Exemple 1.5. Le fichier `deployment/test/replace.server.properties` propre au déploiement de test

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

Exemple 1.6. Le fichier `deployment/preprod/replace.server.properties` propre au déploiement de préproduction

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

Les fichiers de replace ci-dessous permettent de positionner l'adresse du serveur pour chaque environnement.

1.5. Qu'est-ce-qu'un module ?

La notion de module est équivalente à un composant mais est utilisé à l'intérieur d'un projet. La seule différence est la non réutilisabilité. Bien souvent, les modules sont utilisés uniquement pour définir les dépendances binaires et sources d'un projet. Vous pouvez également les utiliser pour définir du code source très spécifique au projet tels que les reprises ou migration de données. Les modules sont stockés dans le fichier `moddep.txt` situé à la racine de l'arborescence du projet.

Exemple 1.7. Un exemple de fichier `moddep.txt`

```
main
```

Cette exemple illustre la déclaration du module `main` au sein d'un projet donné.

1.6. Qu'est-ce-qu'une tâche/activité ?

Nous avons vu en introduction que l'ABS définit des tâches correspondantes à la méthodologie de réalisation de projet. Ainsi il est possible de créer un projet en utilisant la commande `abs new.project` et en valorisant son nom. A chaque exécution de tâche l'ABS chargera le projet, composant ou module sur lequel porte votre action et l'injectera dans un template velocity contenant les instructions Ant nécessaire à la réalisation du traitement. Ainsi, un script Ant sera générée et exécuté ensuite. L'avantage de cette technique est qu'il n'y a pas d'adhérence forte entre les sources d'un composant ou d'un projet avec un outil de build quelconque. Si vous souhaitez changer le langage de scripting "Ant" par une autre moteur, c'est possible ! Il faut considérer l'ABS comme un "shell" qui vous permet d'invoquer différentes commandes dans le contexte d'un projet. De plus, il a l'avantage d'être très paramétrable et dispose d'un gestionnaire de package permettant de mettre à jour les postes utilisateurs, et donc de diffuser très rapidement de nouvelles tâches. Les développeurs ou autres utilisateurs n'ont plus besoin de modifier des scripts Ant, Maven ou autres lors de la réalisation d'un projet !

2

Installation

2.1. Installation basique de l'ABS

ABS requiert java en version 5 ou supérieure.

- Créez un répertoire d'installation (par exemple : ~/abs)
- Téléchargez l'installeur (workshop-repository-manager-0.6.jar [<http://prdownloads.sourceforge.net/abs/workshop-repository-manager-0.6.jar?download>])
- Démarrer l'installation dans le répertoire d'installation (~/abs), à l'aide des commandes suivantes :

```
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
```

Si, vous souhaitez utiliser un proxy, utilisez ces options : `java -jar workshop-repository-manager-0.6.jar repository-url [proxy port [user pwd]]`

- Déclarez la variable ABS_HOME dans votre environnement (ABS_HOME=~/abs)
- Ajoutez le répertoire \$ABS_HOME/tc/ant/bin dans le path de votre système
- Initialiser la chaîne de production :

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

```
cd $ABS_HOME
abs init
```

Répondre aux questions (Quelques valeurs par défaut sont proposé, laissez-les si vous ne savez pas répondre). Cette phase d'initialisation va créer un fichier build.properties dans votre répertoire ABS_HOME. Voici un exemple de build.properties :

```
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. Installer le Plugin Eclipse ABS

Téléchargez la version d'eclipse Europa (3.3) correspondante à votre système : [ici](http://sharengo.org/update/europa) [http://sharengo.org/update/europa] Démarrez eclipse, sélectionnez les menu "Help" -> "Software Updates" -> "Find and Install ..." :

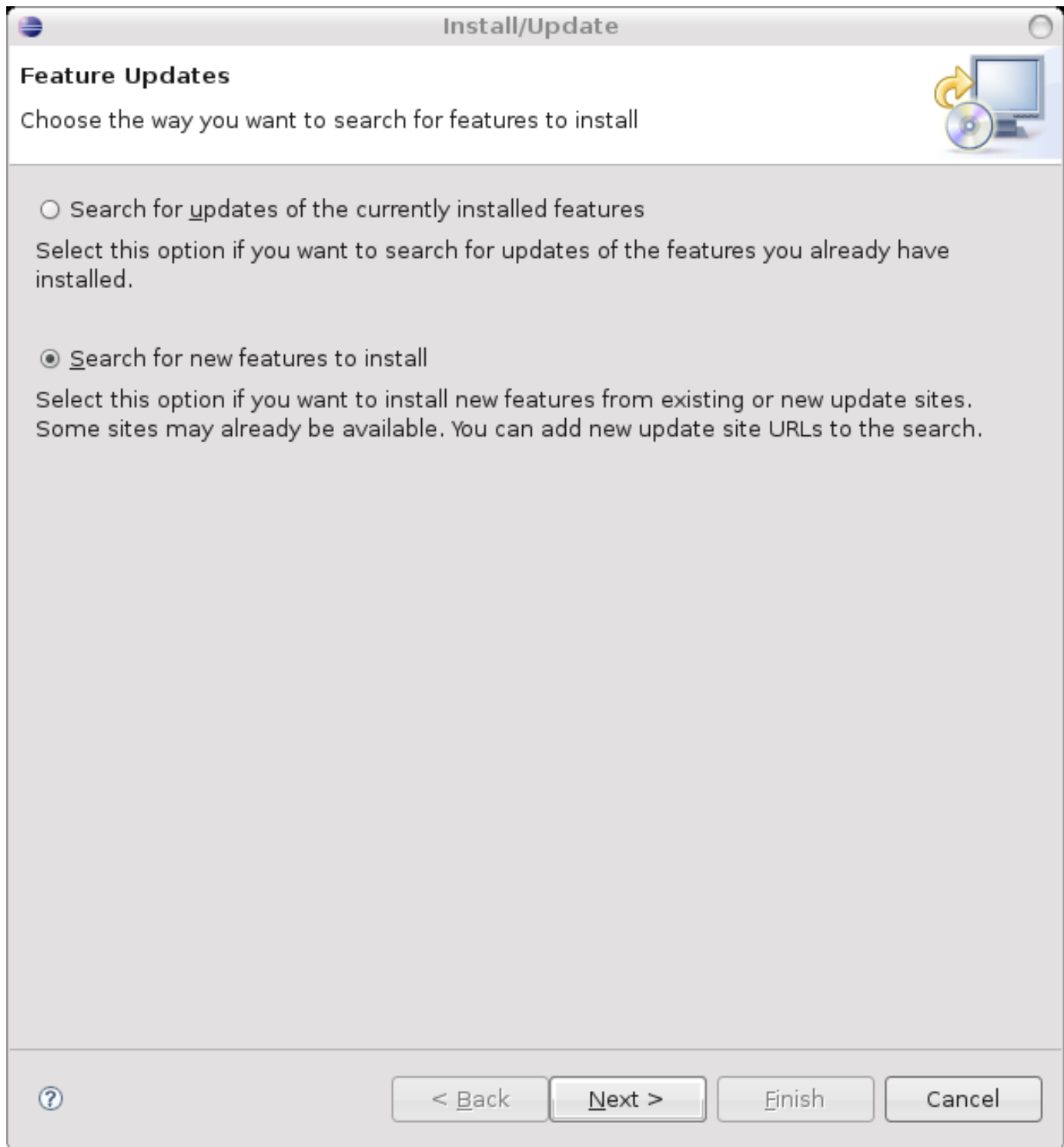


Figure 2.1. Software updates screen shot

Choisissez "Search for news features to install" et cliquez sur "Next" :

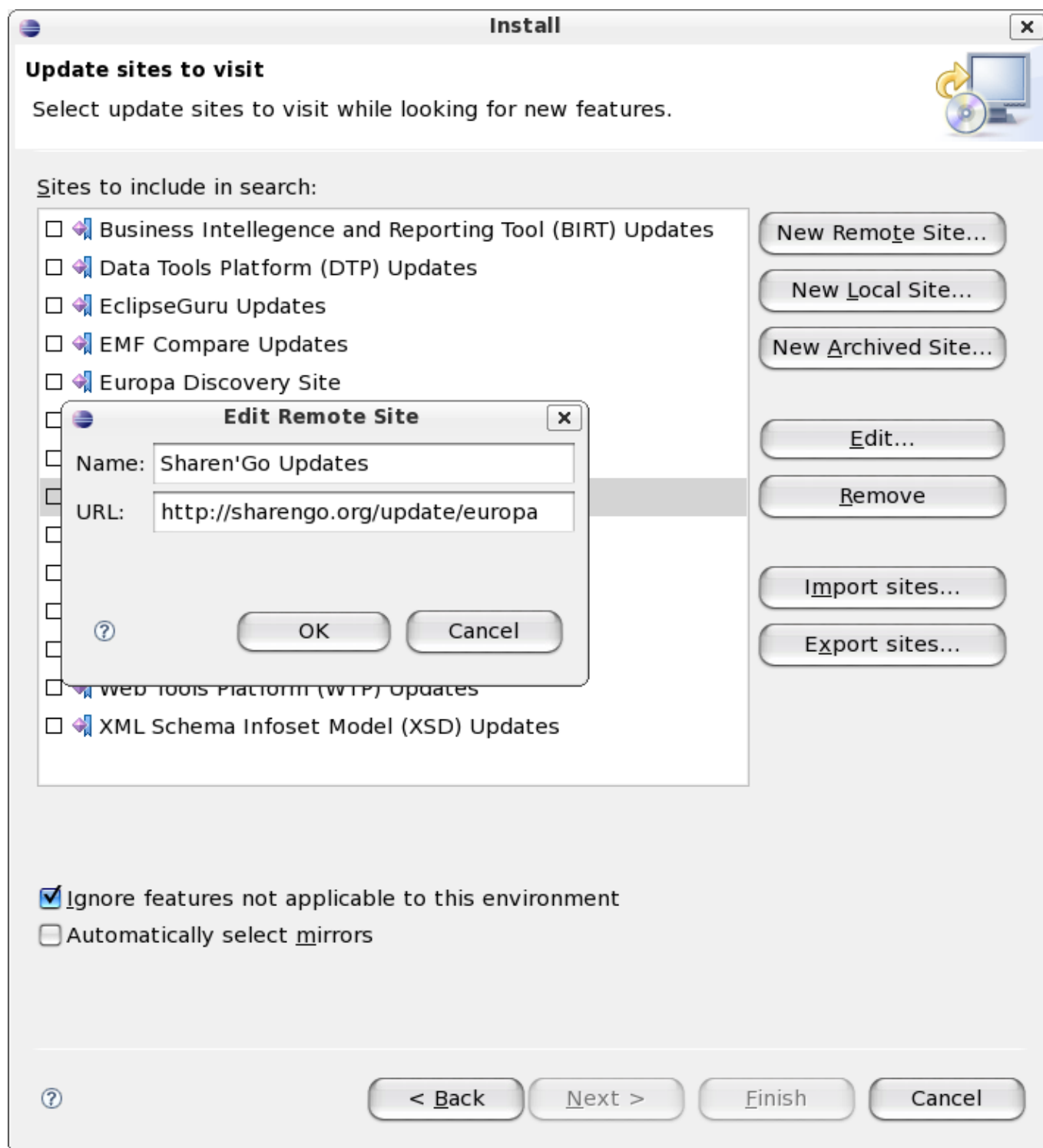


Figure 2.2. Update sites screen shot

A l'aide du bouton "New Remote Site", ajoutez le site de mise à jour Sharen'Go : <http://sharengo.org/update/europa>, puis sélectionnez-le et cliquez sur "Finish". Sélectionnez ensuite les outils Sharen'Go dans la liste des fonctionnalités offertes et cliquez sur Next :

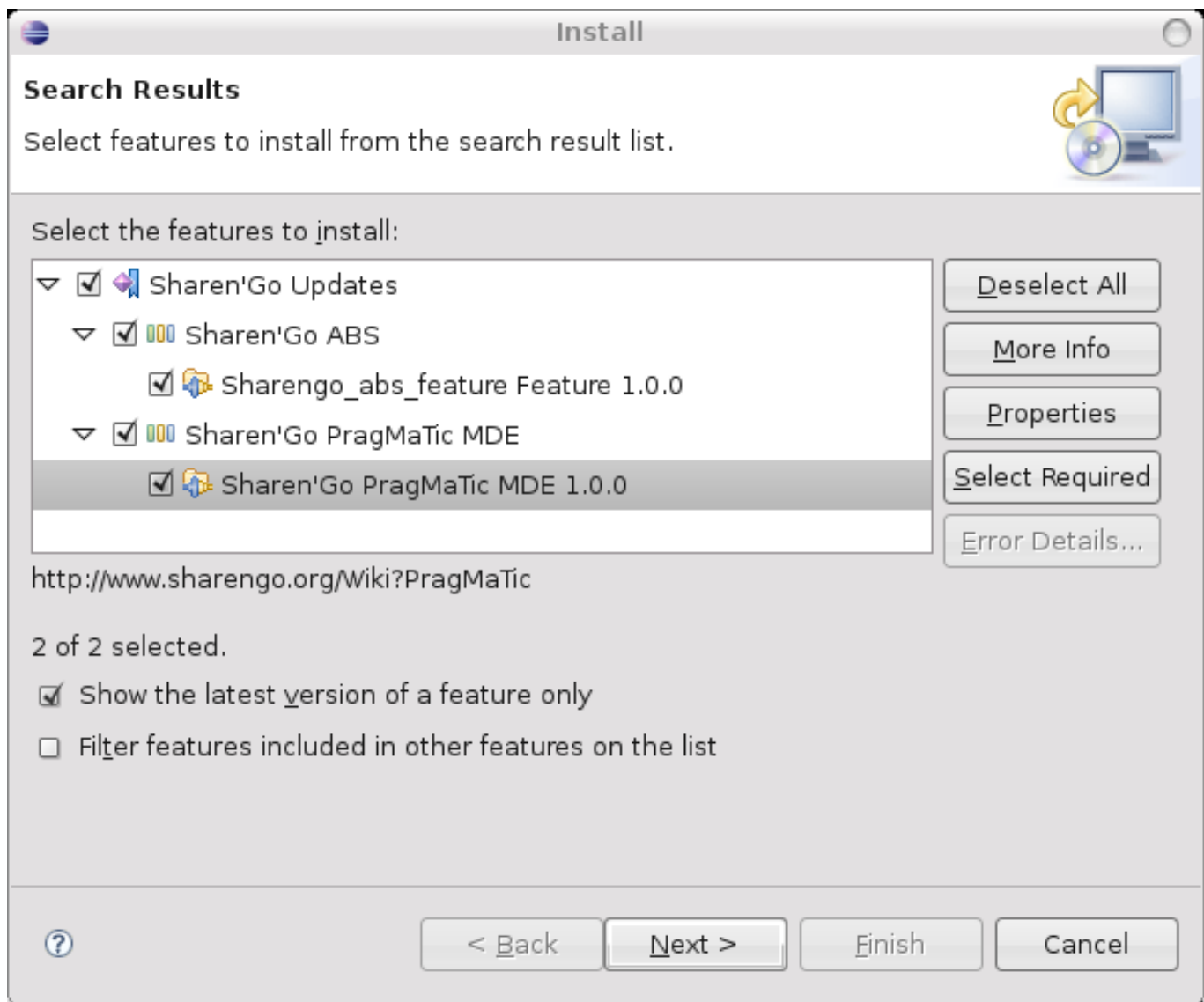


Figure 2.3. Install features screen shot

Accepter les licences et cliquez sur "Finish"

3

Utilisation en ligne de commande

Ce chapitre ne se veut pas exhaustif, il présente les principales commandes utiles. Vous pouvez consulter la liste complète à l'aide de la commande suivante :

```
$> abs -p
```

Un exemple :

```
$> 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. Qu'est-ce qu'un workspace

Un workspace est un espace de travail permettant de cloisonner vos contextes de travail. Il définit un espace physique de stockage d'un projet et de l'ensemble de ses composants dépendants. Une bonne pratique consiste à utiliser un workspace par version de projet. Ainsi lorsque vous réalisez une phase donnée d'un projet, vous devez créer un workspace. Ensuite lorsque vous commencerez une seconde phase, créez un autre workspace afin d'isoler la phase 2 de la phase 1. Ainsi si vous devez revenir sur la première, vous pourrez aisément relivrer une version n'intégrant que votre correction et non les évolutions liées à la phase 2. Vous trouverez ci-après un exemple de création de workspace.

```
$> 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. Créer un projet

Tout d'abord, vous devez vous placer dans le workspace précédemment créé.

3.2.1. Créer un nouveau projet

La commande "new.project" permet de créer un nouveau projet en local (local signifie que le projet n'existe pas dans le référentiel).

```
$> 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. Récupérer un projet existant

Vous pouvez également récupérer un projet existant à partir du référentiel à l'aide de la commande "new.scm.project" (la configuration des différents référentiels utilisables est stockée dans le fichier `$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. Récupérer un projet existant dans une version donnée

Il existe une variante de la commande précédente permettant de récupérer un projet à partir du référentiel et dans une version donnée. Cette commande est utile principalement avec CVS, quant à subversion la commande précédente suffit puisque le numéro de version (le tag) est porté par le nom du projet, par exemple : `project2/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. Créer un composant

Tout d'abord, vous devez vous placer dans le workspace précédemment créé.

3.3.1. Créer un nouveau composant

La commande "new.component" permet de créer un nouveau projet en local (local signifie que le composant n'existe pas dans le référentiel).

```
$> 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. Récupérer un composant existant

Vous pouvez également récupérer un projet existant à partir du référentiel à l'aide de la commande "new.scm.component" :

```
$> 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. Récupérer un composant existant dans une version donnée

Il existe une variante de la commande précédente permettant de récupérer un composant à partir du référentiel et dans une version donnée. Cette commande est utile principalement avec CVS, quant à subversion la commande précédente suffit puis que le numéro de version (le tag) est porté par le nom du composant, par exemple : `component2/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. Compiler

La commande "lib.local" permet de compiler un composant ou un projet. Cette opération construit 3 archives *.jar contenant les fichiers *.class correspondant à la version binaire des fichiers source *.java. Vous pouvez également la commande "lib:all" qui propage l'opération de compilation sur l'ensemble des composants utilisés par la ressource courante. Les 3 archives résultantes sont :

- `NOM_DU_COMPOSANT.jar` : archive correspondant aux sources stockés dans le répertoire common
- `NOM_DU_COMPOSANT-client.jar` : archive correspondant aux sources stockés dans le répertoire client
- `NOM_DU_COMPOSANT-server.jar` : archive correspondant aux sources stockés dans le répertoire server

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

...
BUILD SUCCESSFUL
```

3.5. Créer un déploiement

La commande "deploy:new" permet de créer un nouvel environnement cible. Elle ajoute un répertoire portant le nom du déploiement dans le répertoire `deployment` situés à la racine du projet. Elle crée également les fichiers `replace.server.properties` et `replace.client.properties` permettant de variabiliser les fichiers de configuration utilisés lors de la construction des livrables relatif à cet environnement.

```
$> 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. Construire un livrable projet

La commande "deploy:build.war" construit une archive `WAR` permettant de déployer le projet sur un serveur d'application tel que Apache-Tomcat.

La commande "deploy:build.cli" construit une archive `.tar.gz` permettant de lancer l'application à l'aide d'une exécutable en ligne de commande.

Ces 2 commandes construisent une archive par déploiement spécifiés dans le projet en configurant leurs spécifités à l'aide des fichiers de `replace`.

4

Utilisation sous Eclipse

4.1. Créer un composant

A l'aide du menu : "New" -> "Project" -> dans le noeud ABS sélectionnez "New Component"

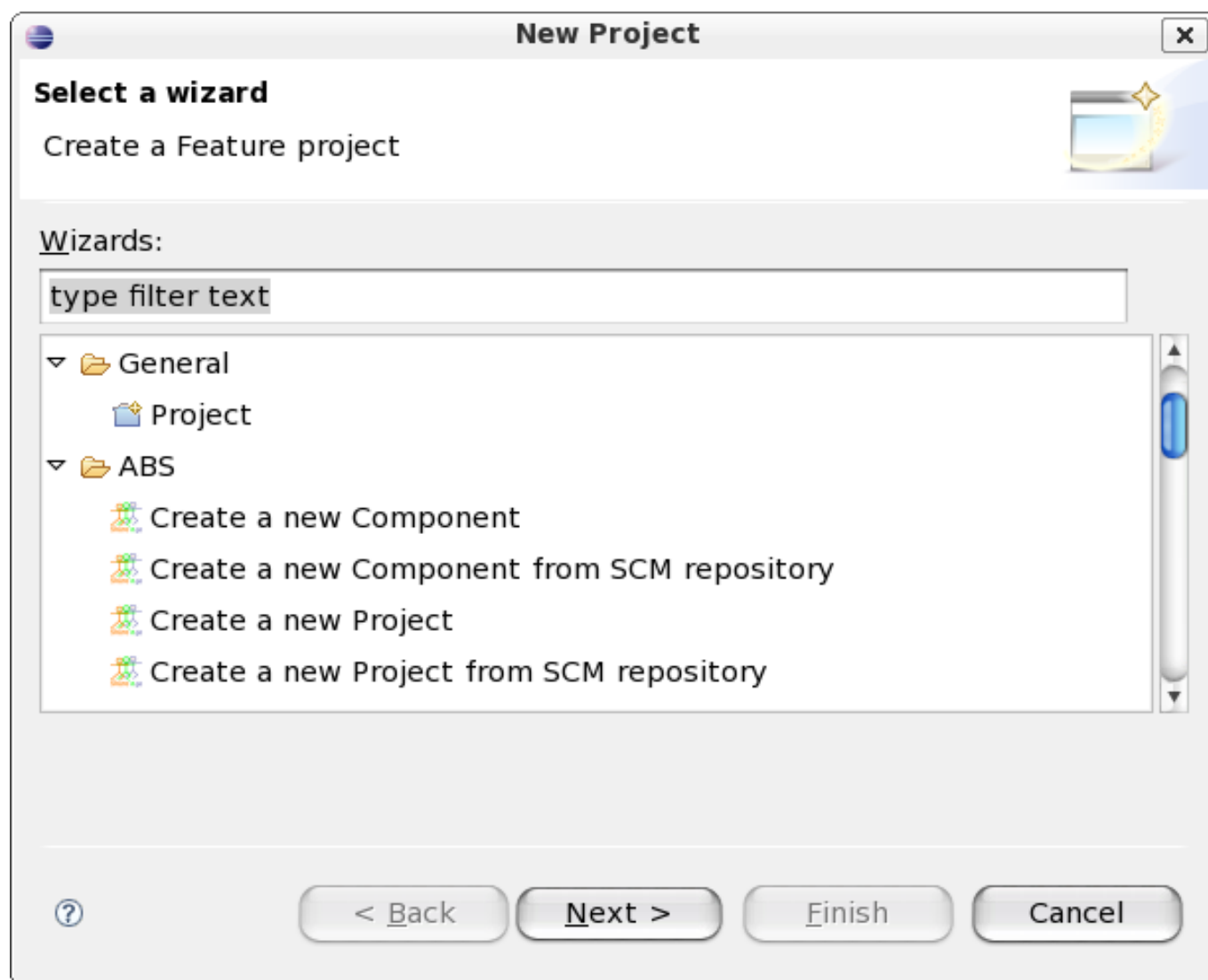


Figure 4.1. Capture de l'écran de création de composant

Le plug-in eclipse interagit avec l'ABS installé sur votre système, il est donc nécessaire d'indiquer l'emplacement de l'ABS_HOME ainsi que celui du workspace ABS. Si cette information n'est pas renseignée dans les préférences

ABS d'eclipse (menu "Window" -> "Preferences ..." -> rubrique "ABS Preference") l'assistant suivant vous sera proposer et se chargera de les mettre à jour :

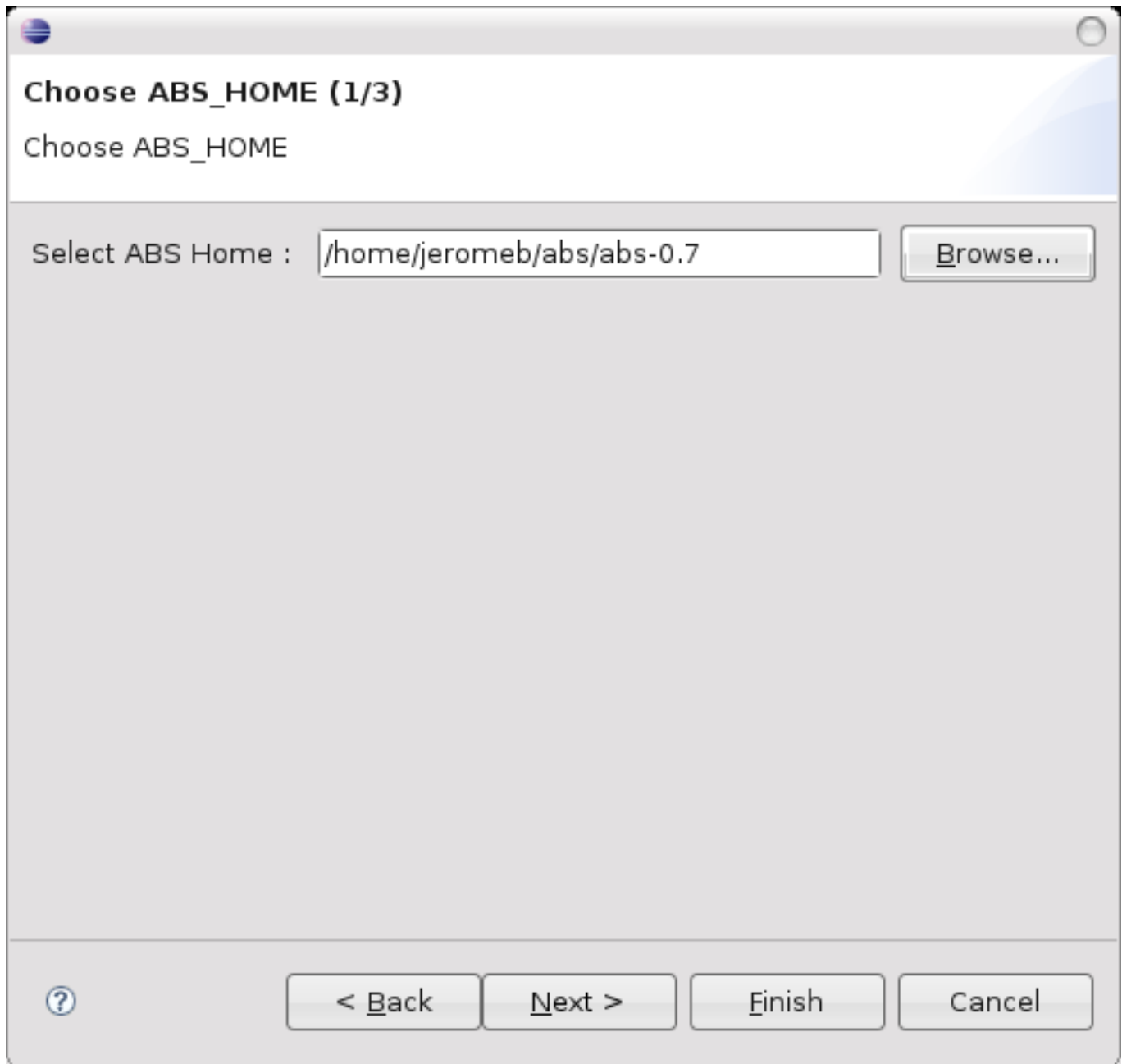


Figure 4.2. Capture d'écran de sélection de l'ABS HOME

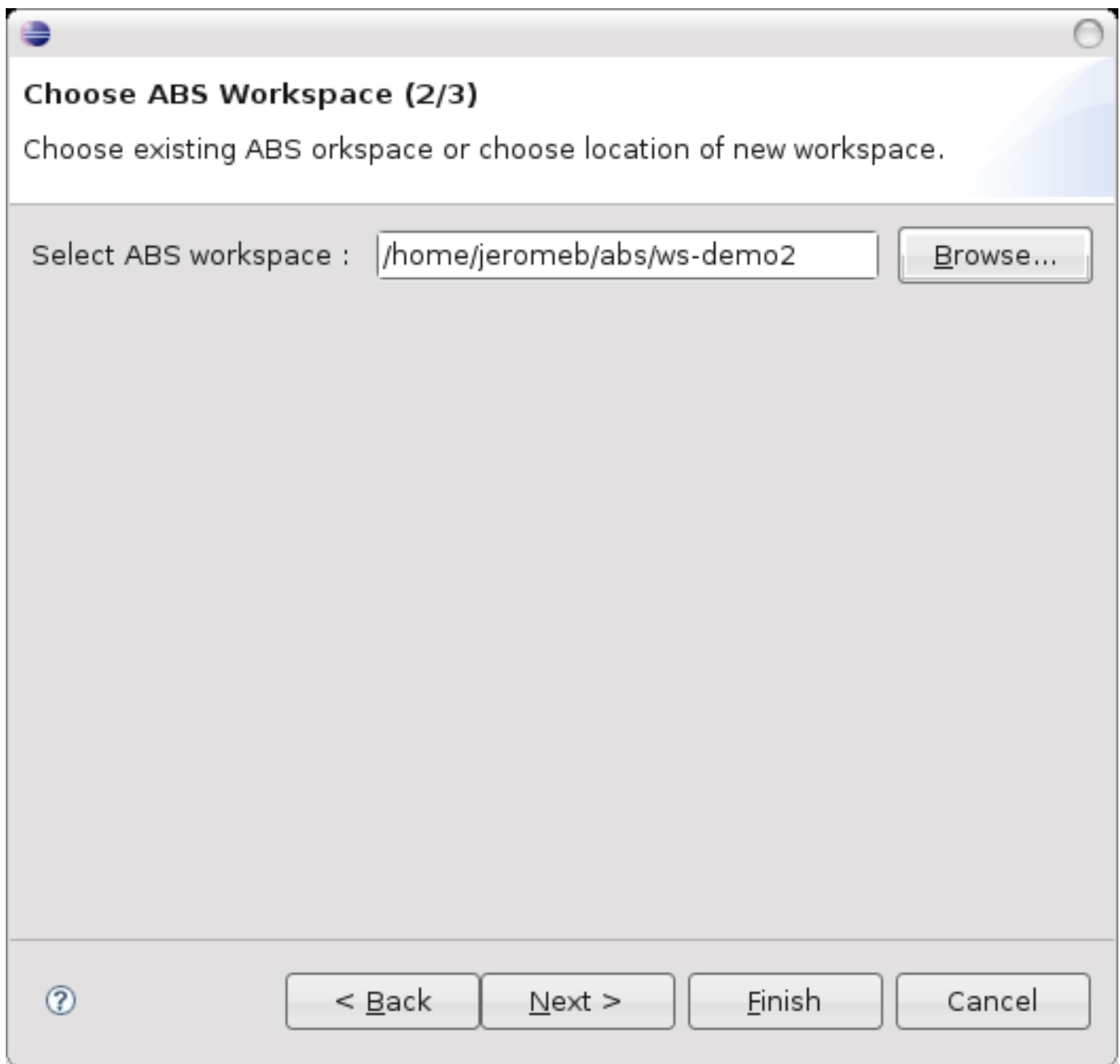


Figure 4.3. Capture d'écran de sélection d'un workspace ABS

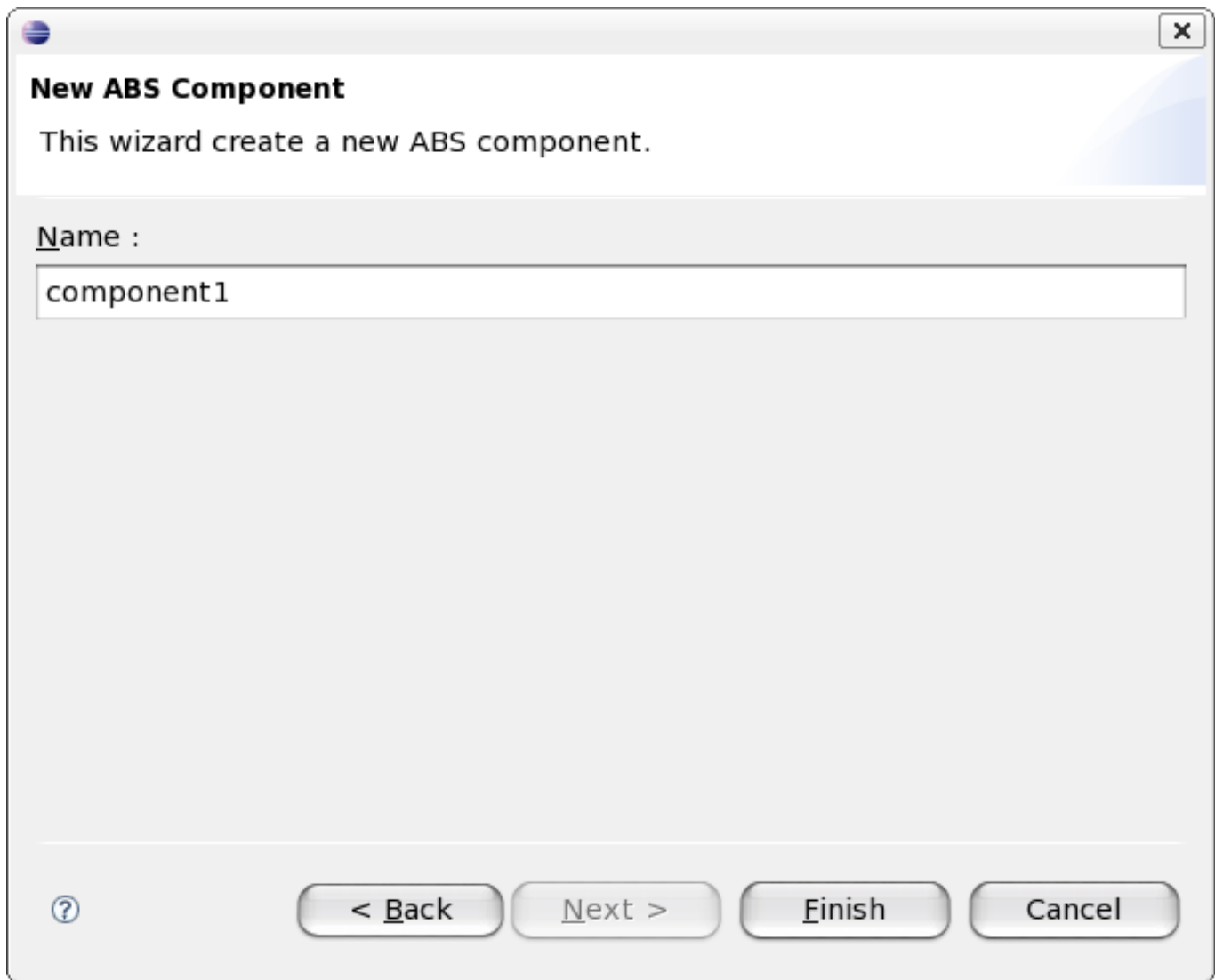


Figure 4.4. Capture d'écran de saisi des informations du composant

4.2. Création d'un projet

A l'aide du menu : "New" -> "Project" -> dans le noeud ABS sélectionnez "New Project"

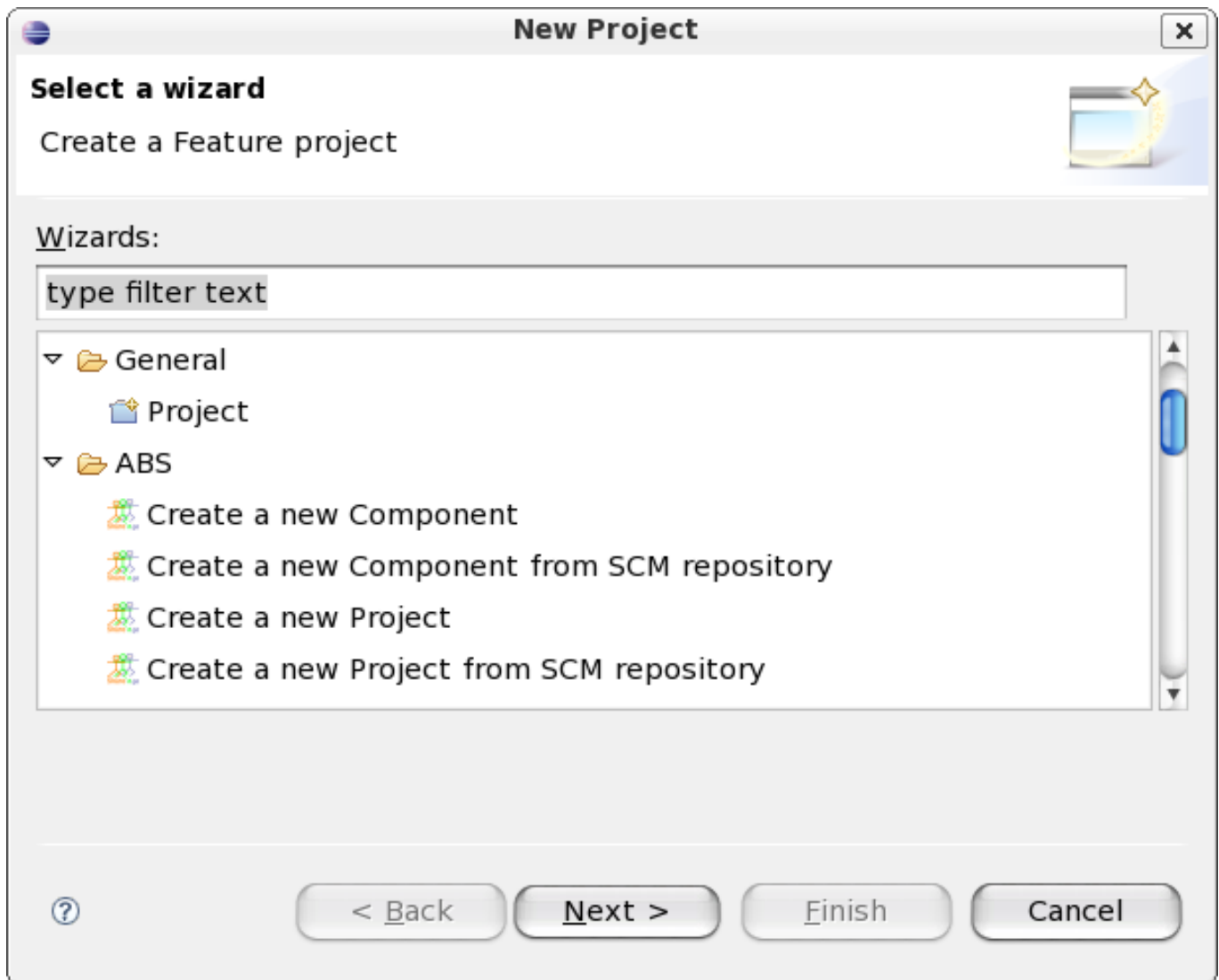


Figure 4.5. Capture de l'écran de création de projet

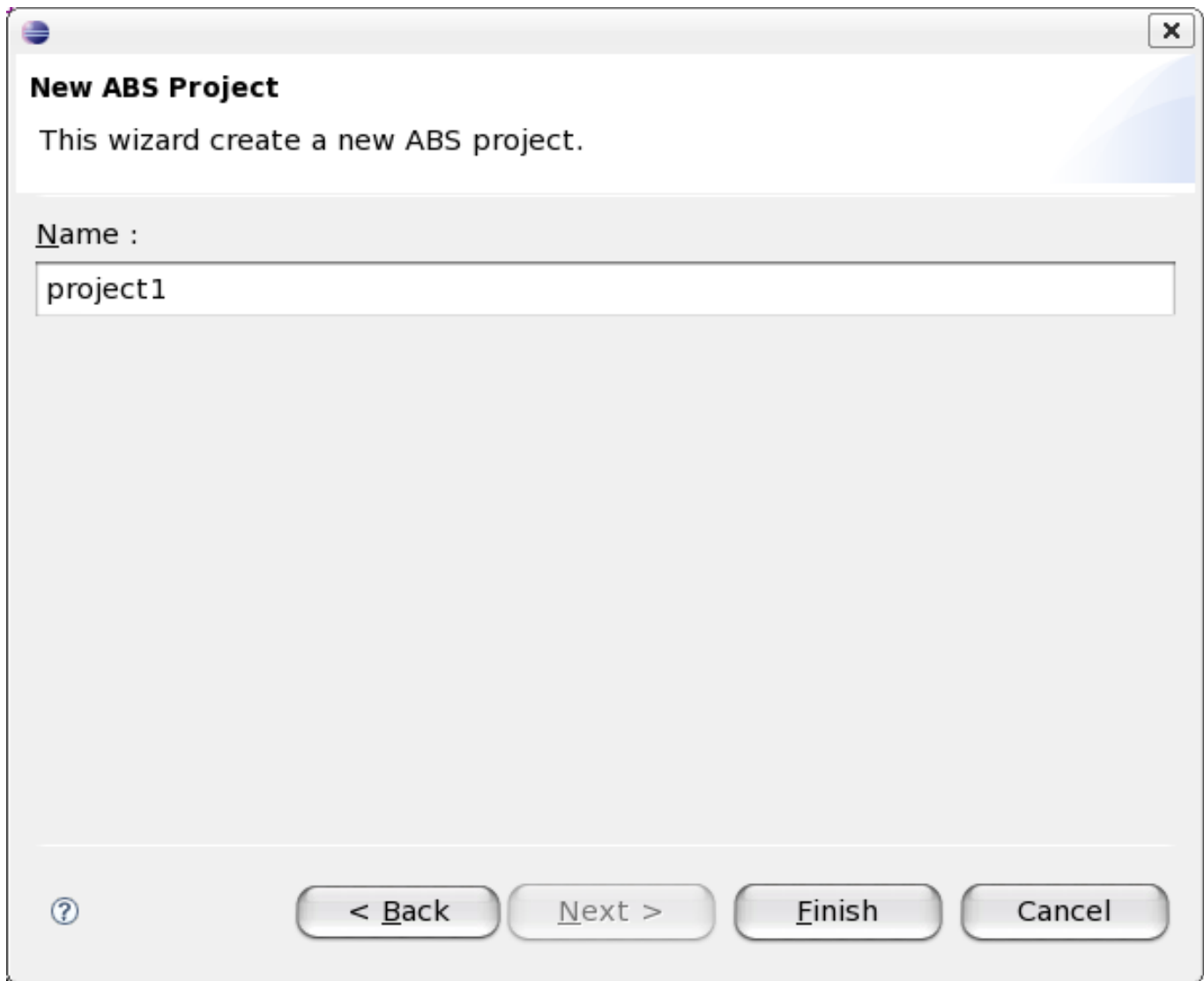


Figure 4.6. Capture d'écran de saisi des informations du projet

Lorsque vous créez un projet, l'assistant ajoute automatiquement un module nommé "main" et un déploiement nommée "localhost". Vous trouverez dans la vue "PackageExplorer" ci-dessous le module "project1_main", ainsi le déploiement "project1_localhost_server".

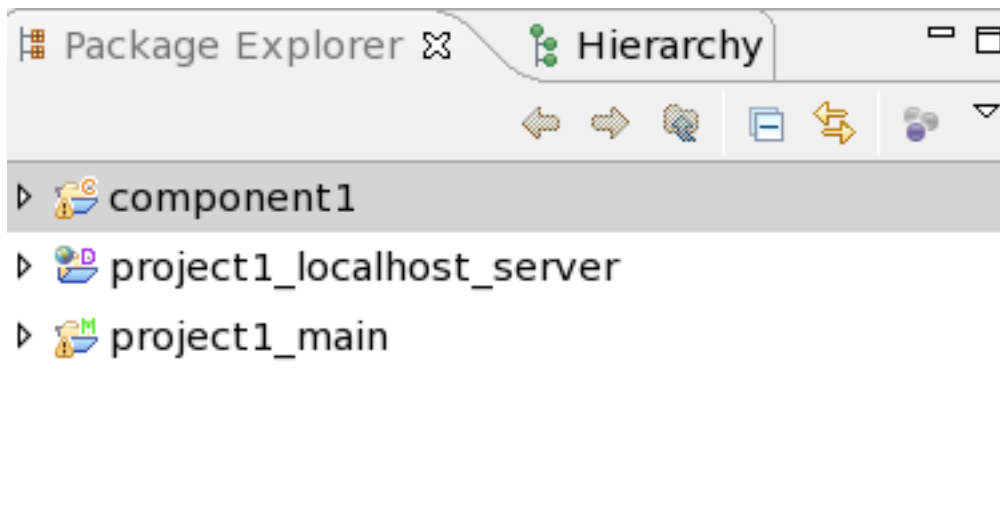


Figure 4.7. Capture de la vue package explorer permettant de visualiser les ressources du projet

4.3. Créer un modèle de conception

Vous pouvez créer un modèle de conception UML dans un composant ou un module de projet.

4.3.1. Modéliser en UML

Positionnez-vous dans la répertoire `server/model/conception` et créez un modèle nommé `conception.uml` à l'aide du menu "New" -> "Other..." -> "SharenGo Unified Modeling" -> "Step 3 - New Conception Model" et créez le fichier `conception.uml` :

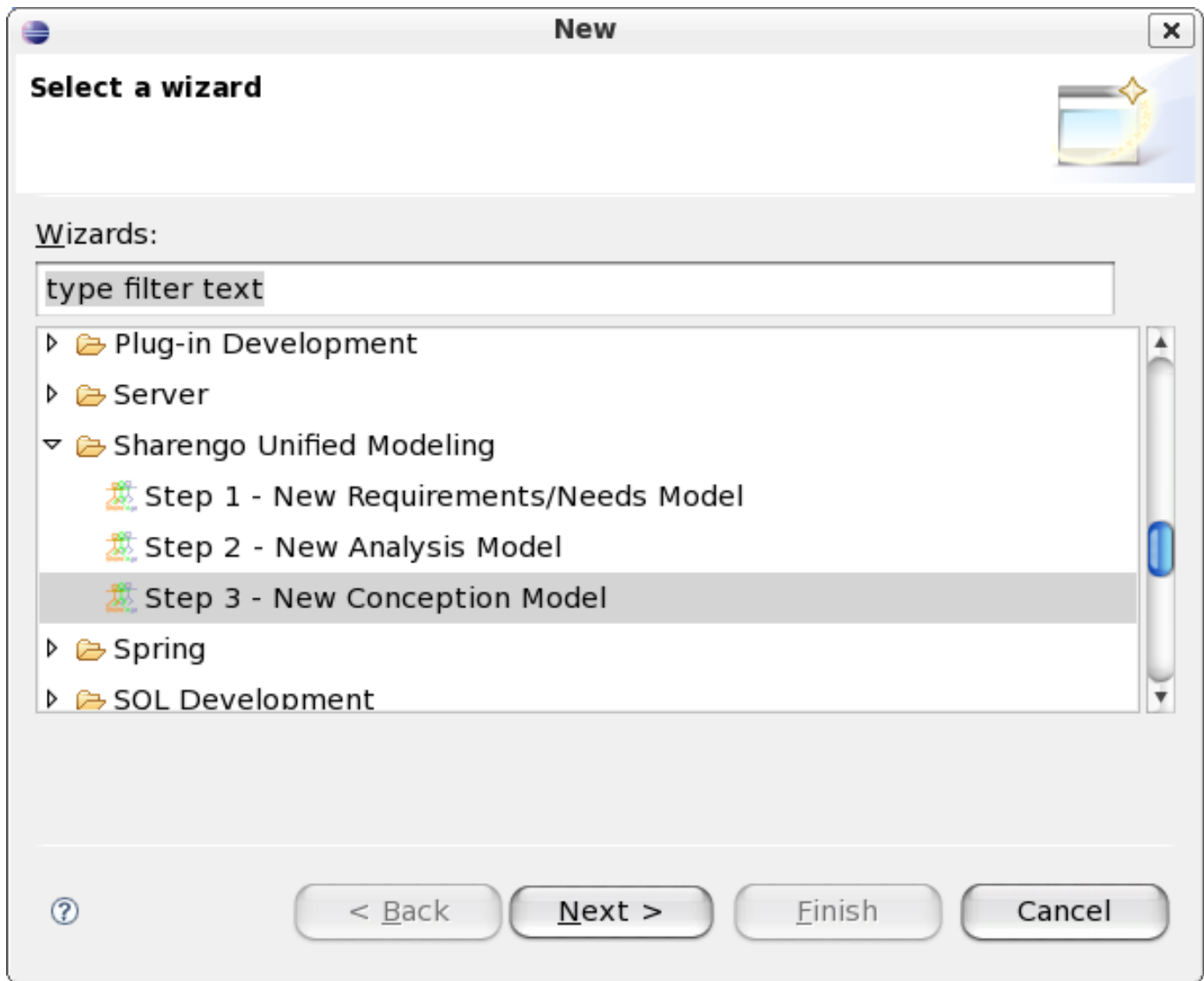


Figure 4.8. Capture d'écran de la création de modèle de conception

Cette commande crée trois fichiers :

- `conception.uml` : ce fichier contient les informations structurelles du modèle,
- `conception.uml-di` : ce fichier contient les diagrammes, il utilise le premier pour référencer les éléments utilisés au sein des diagrammes,
- `conception.properties` : ce fichier permet de paramétrer la stratégie de génération, les valeurs par défaut sont suffisantes pour une utilisation courante.

Note

Il est impératif de nommer le modèle `org::sharengo::nom_composant` à l'aide de la fenêtre propriétés. (les `::` étant le séparateur de package utilisé en langage UML) Ce nom de modèle permettra de déterminer le nom du paquetage contenant les fichiers générés.

4.3.2. L'architecture logique

Pour modéliser les divers composants nous avons définis une architecture logique qui correspond à un PIM (Platform Independant Model) selon la dénomination MDA. Le schéma ci-dessous illustre les différentes couches de cette architecture basée sur le patron de conception MVC (Model View Controller) :

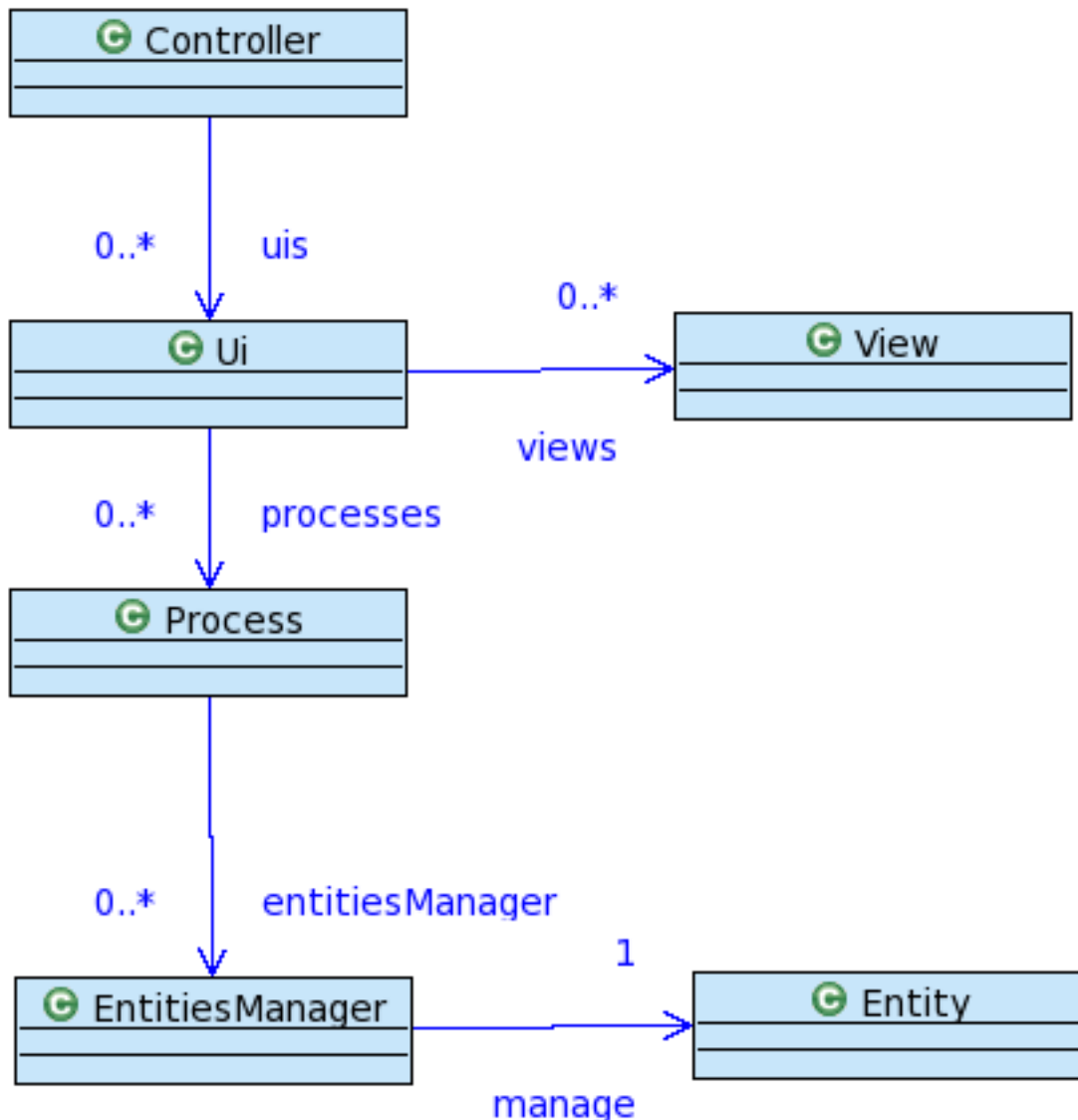


Figure 4.9. Schéma d'architecture logique

Ce modèle d'architecture peut être considéré comme un métamodèle puisqu'il sera instancié au sein de chaque composant et module. Pour des raisons historiques, nous avons choisi d'instancier ce modèle à l'aide du langage UML et non à l'aide d'un DSM. (Domain Specific Modeler) Nous avons donc réalisé un profil UML pour étendre UML afin d'y intégrer les concepts de notre métamodèle. Pour ce faire nous avons définis une batterie de stéréotypes et propriétés que nous allons décrire dans la section suivante.

4.3.2.1. Les stéréotypes

Les stéréotypes permettent d'apposer une sémantique particulière aux éléments UML. Appliquer un stéréotype dans l'outil de modélisation TopCased :

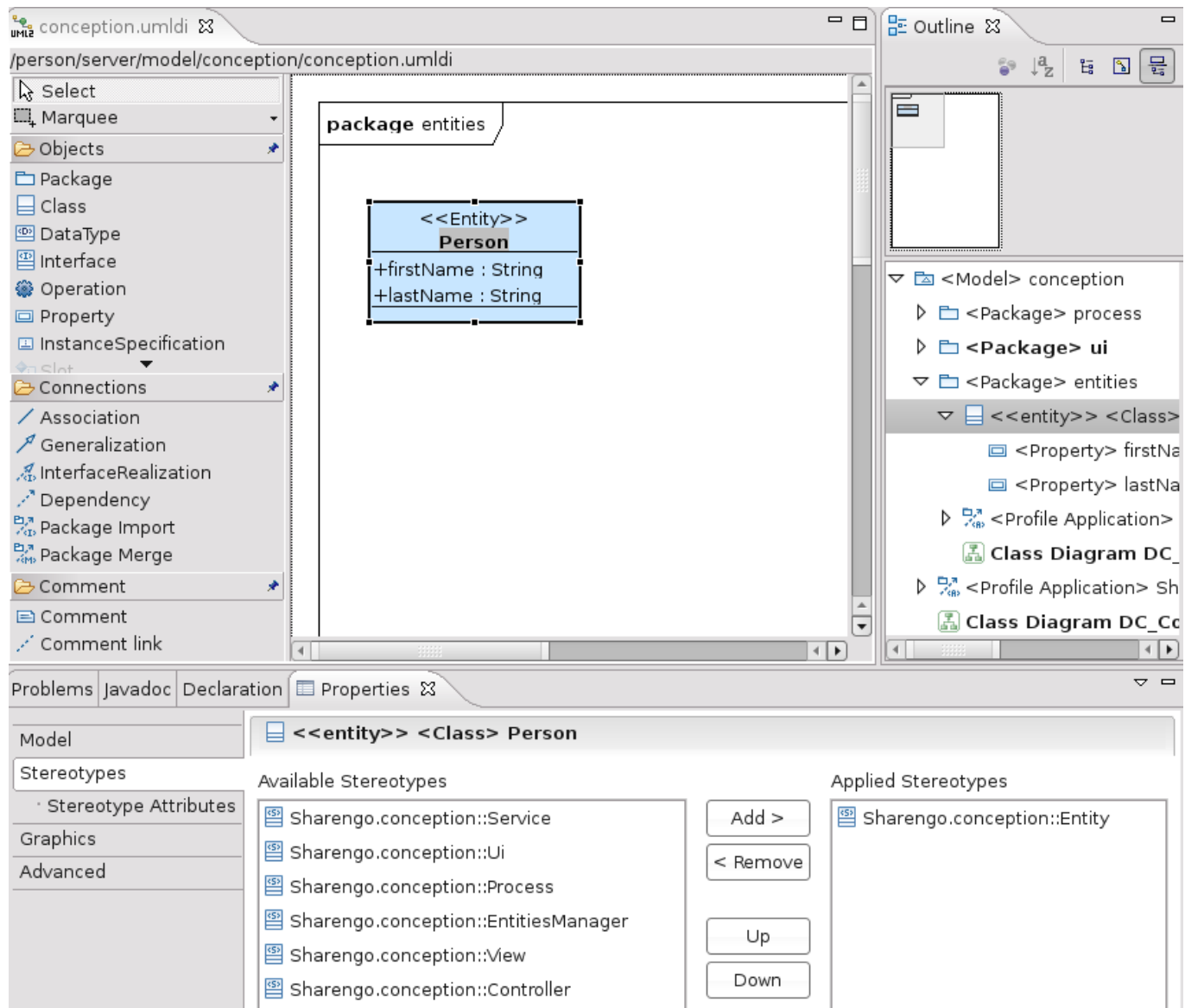


Figure 4.10. Application d'un stéréotype

Un stéréotype peut définir des propriétés que vous pouvez valoriser dans l'onglet "Stereotype Attributes".

4.3.2.2. Le stéréotype <<Entity>>

Définition. Ce stéréotype est utilisé pour spécifier les objets du domaine métier (persités par l'application).

Propriétés. Aucune.

Contexte d'utilisation.

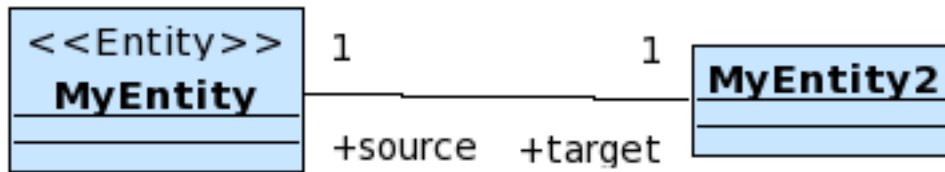


Figure 4.11. Association entre 2 entités

4.3.2.3. Le stéréotype <<Dto>>

Définition. Ce stéréotype est utilisé pour spécifier les objets de transfert. Le patron de conception DTO (Data Transfert Object) est utilisé afin de fournir une couche de médiation entre la couche métier et la couche IHM (Interface Homme Machine).

Propriétés. Aucune.

Contexte d'utilisation. Souvent utilisé en tant que paramètre d'une opération.

4.3.2.4. Le stéréotype <<EntitiesManager>>

Définition. Ce stéréotype est utilisé pour spécifier les objets d'accès aux données. Il représente le patron de conception DAO (Data Access Object). Ce type de classe assure la sauvegarde et la restauration des données définies dans les entités métiers.

Propriétés. Aucune.

Dépendances possibles. TODO ajouter un screenshot

4.3.2.5. Le stéréotype <<Process>>

Définition. Ce stéréotype est utilisé pour définir un service métier, un traitement métier.

Propriétés. Aucune.

Contexte d'utilisation. Un process peut utiliser d'autres process, ou des EntitiesMgr.

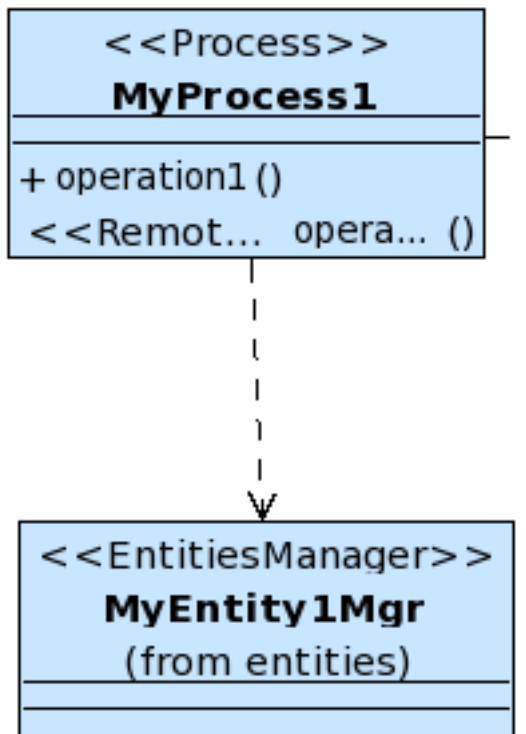


Figure 4.12. Cas d'utilisation d'un EntityManager

4.3.2.6. Le stéréotype <<Controller>>

Définition. Ce stéréotype est utilisé pour définir l'aiguillage des actions utilisateurs. Il décode les informations transmises et les transmet à l'Ui.

Propriétés. Aucune.

Contexte d'utilisation.

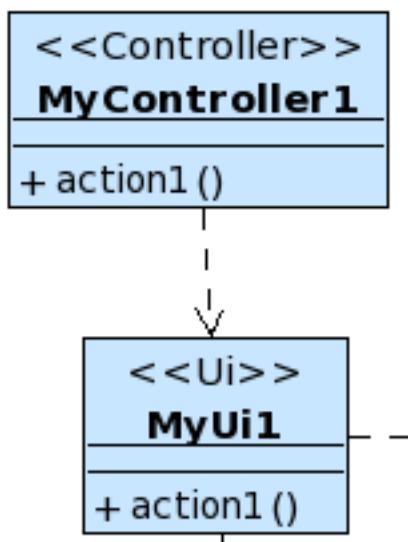


Figure 4.13. Cas d'utilisation d'une Ui

4.3.2.7. Le stéréotype <<Ui>>

Définition. Ce stéréotype est utilisé pour représenter les classes chargées d'orchestrer les appels de services métiers et de transmettre les informations résultantes vers une vue donnée.

Propriétés. Aucune.

Contexte d'utilisation. Une Ui peut utiliser d'autres Ui, faire appels à des Process et utiliser des View.

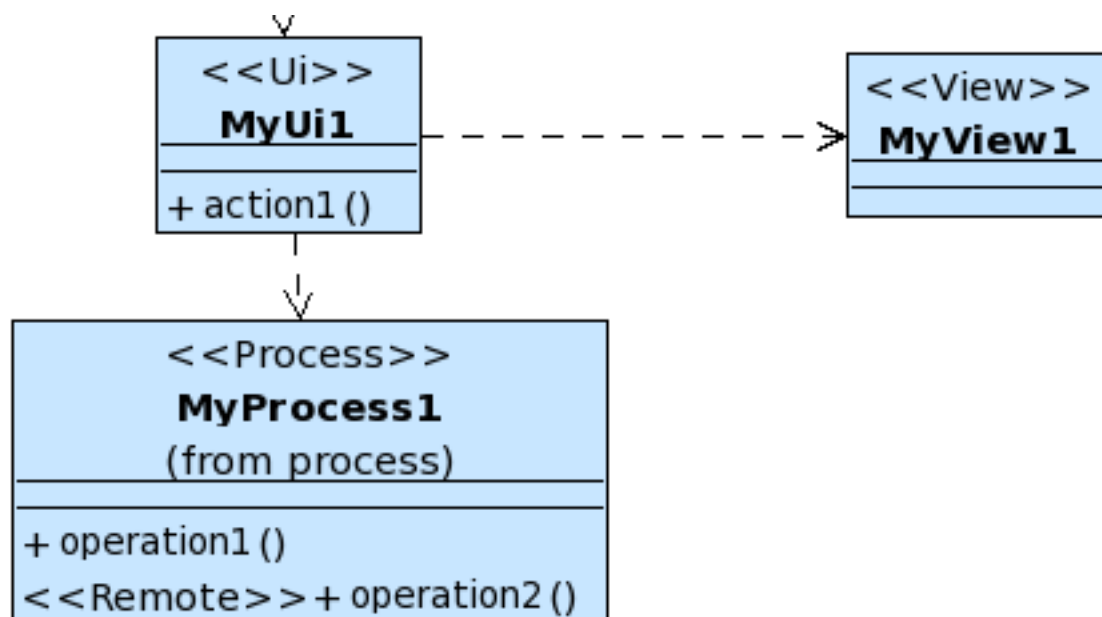


Figure 4.14. Cas d'utilisation d'une Ui

4.3.2.8. Le stéréotype <<View>>

Définition. Ce stéréotype est utilisé pour définir la notion d'écran utilisateur.

Propriétés. Aucune.

Contexte d'utilisation. Une Vue est simplement utilisée par des Ui.

4.3.2.9. Le stéréotype <<Remote>>

Définition. Ce stéréotype permet de représenter qu'une opération est joignable de manière distante. Dans le cas d'une Ui pour mettre en œuvre une communication de type Ajax. Ou de type Web Service si elle est appliquée à une opération d'un Process.

Propriétés. TODO

4.3.2.10. Le stéréotype <<Transactional>>

Définition. Ce stéréotype permet de représenter l'aspect transactionnel d'une opération d'un Process.

Propriétés. Aucune.

4.3.2.11. Le stéréotype <<Config>>

Définition. Ce stéréotype permet de représenter les classes définissant les variables de configuration. Pour définir l'adresse d'un serveur smtp, il est nécessaire de définir un attribut smtpServer de type String dans une classe portant ce stéréotype.

Propriétés. Aucune.

Contexte d'utilisation. Les classes Config peuvent être utiliser par la plupart des classes de l'architecture.

4.4. Générer du code

Nous utilisons l'outil de transformation modèle vers texte Acceleo pour générer le code source à partir du modèle UML ci-dessus. Pour lancer une transformation il est nécessaire de définir une chaîne de génération. Dans notre cas, nous allons définir une chaîne en appelant une autre implémentant une architecture donnée.

Créer la chaîne `conception.launch` à l'aide du menu "New" -> "Others" -> "Acceleo" -> "Module Launcher" :

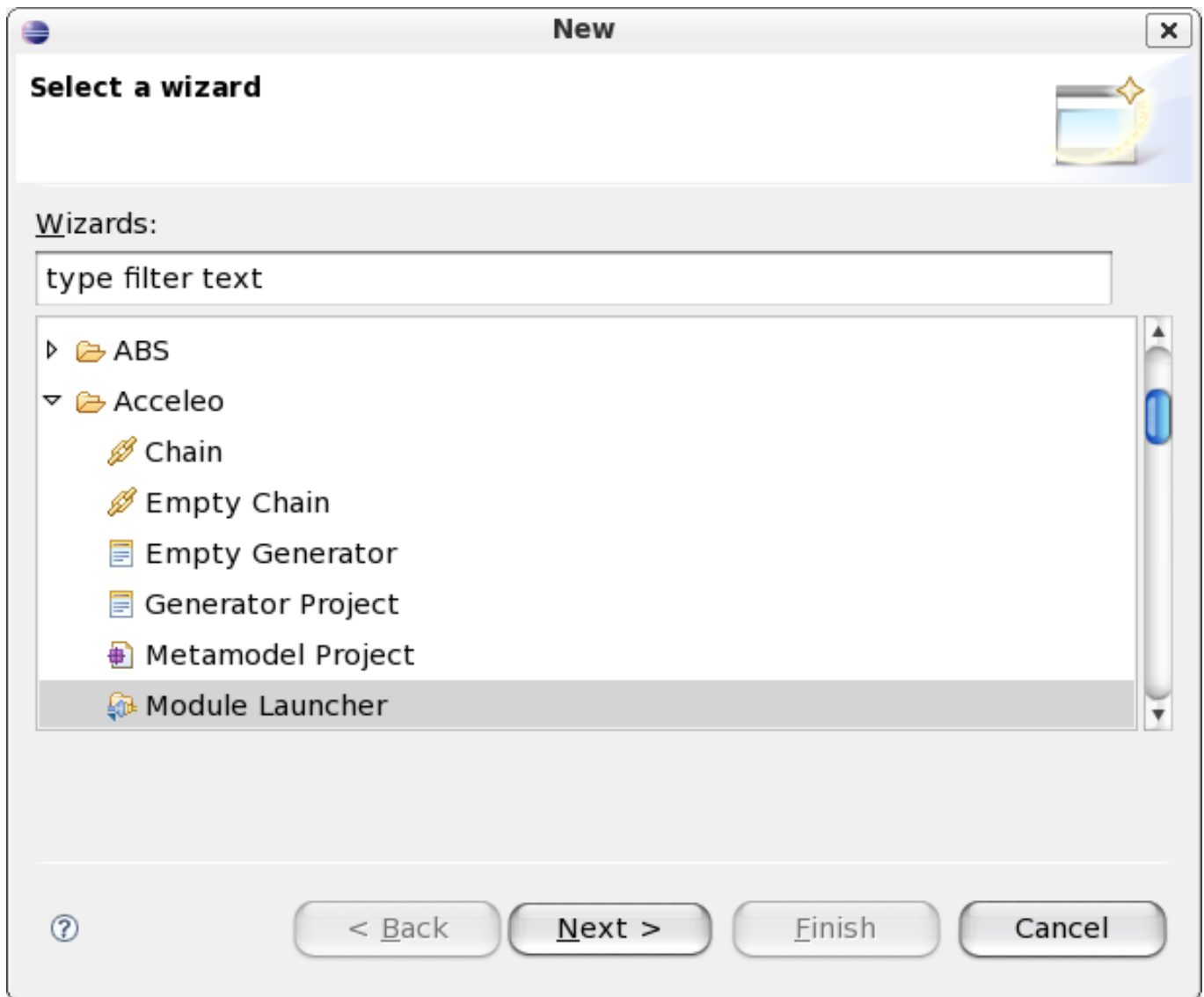


Figure 4.15. Capture de l'écran de l'assistant ModuleLauncher, étape 1

Choisir la chaîne nommée : "Java Generator for Spring / Hibernate / Velocity Architecture" :

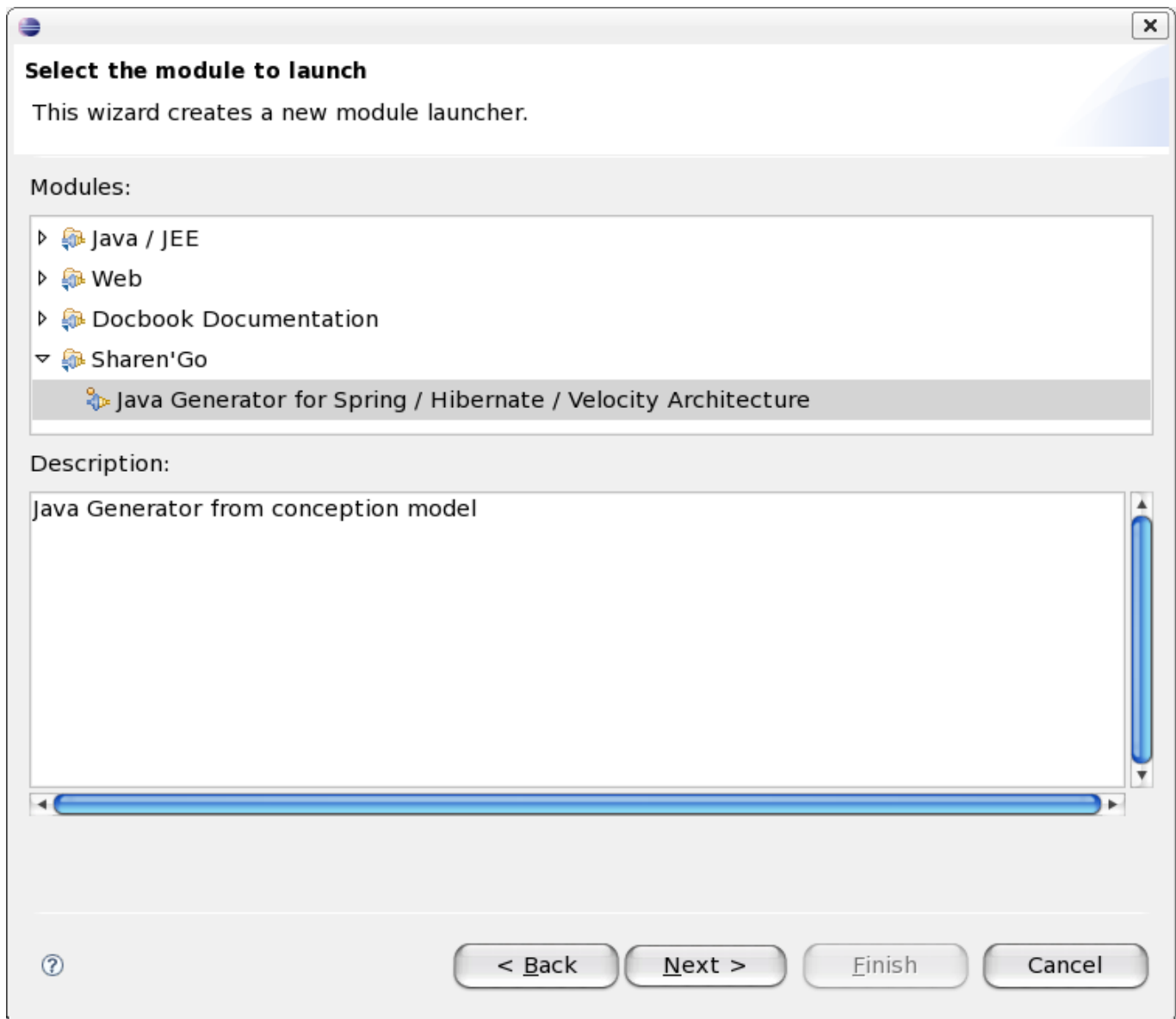


Figure 4.16. Capture de l'écran de l'assistant ModuleLauncher, étape 2

Cliquer sur "Next", nommer la chaîne `conception.chain` :

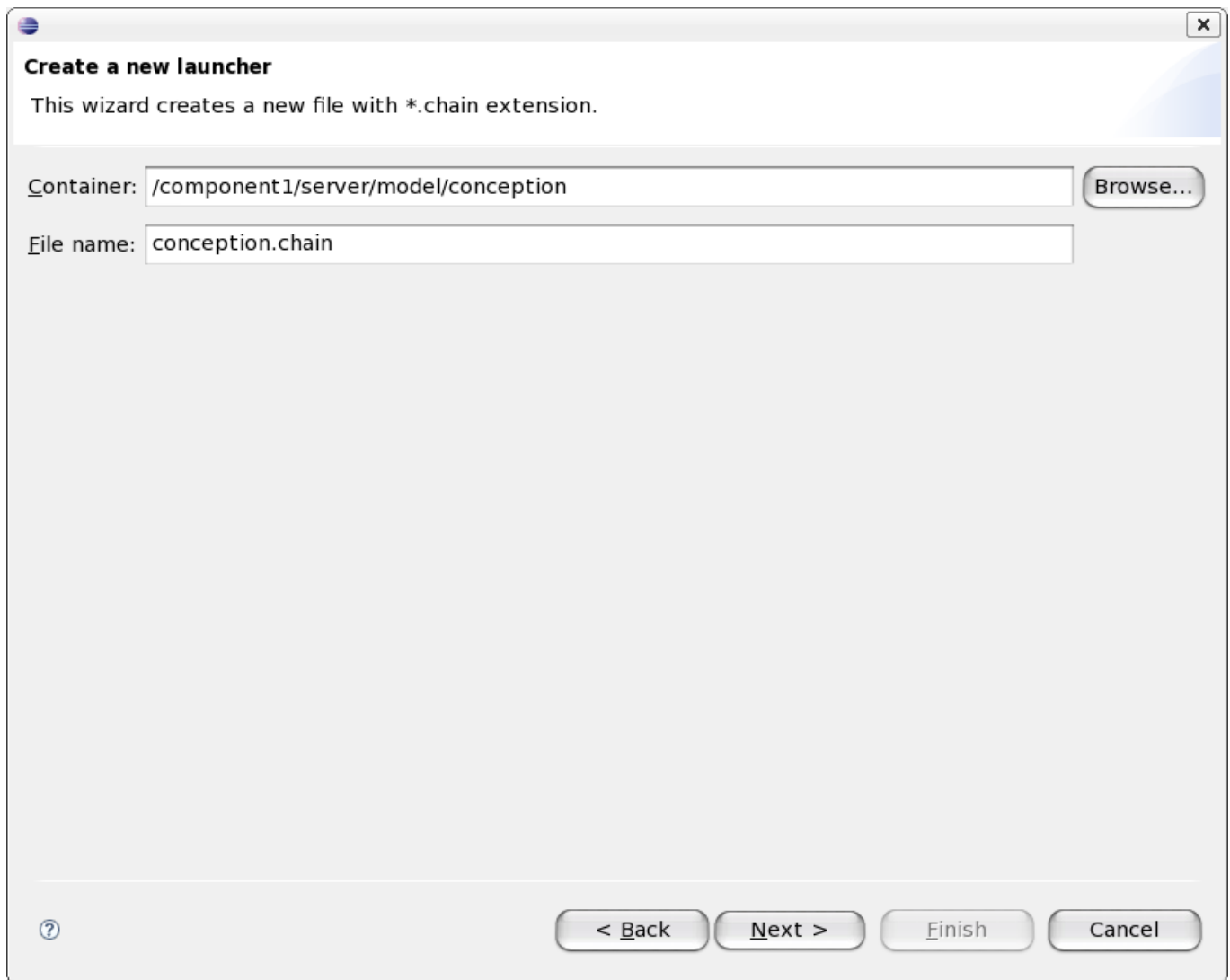


Figure 4.17. Capture de l'écran de l'assistant ModuleLauncher, étape 3

Cliquer sur "Next", sélectionner le modèle UML source (soit conception.uml).

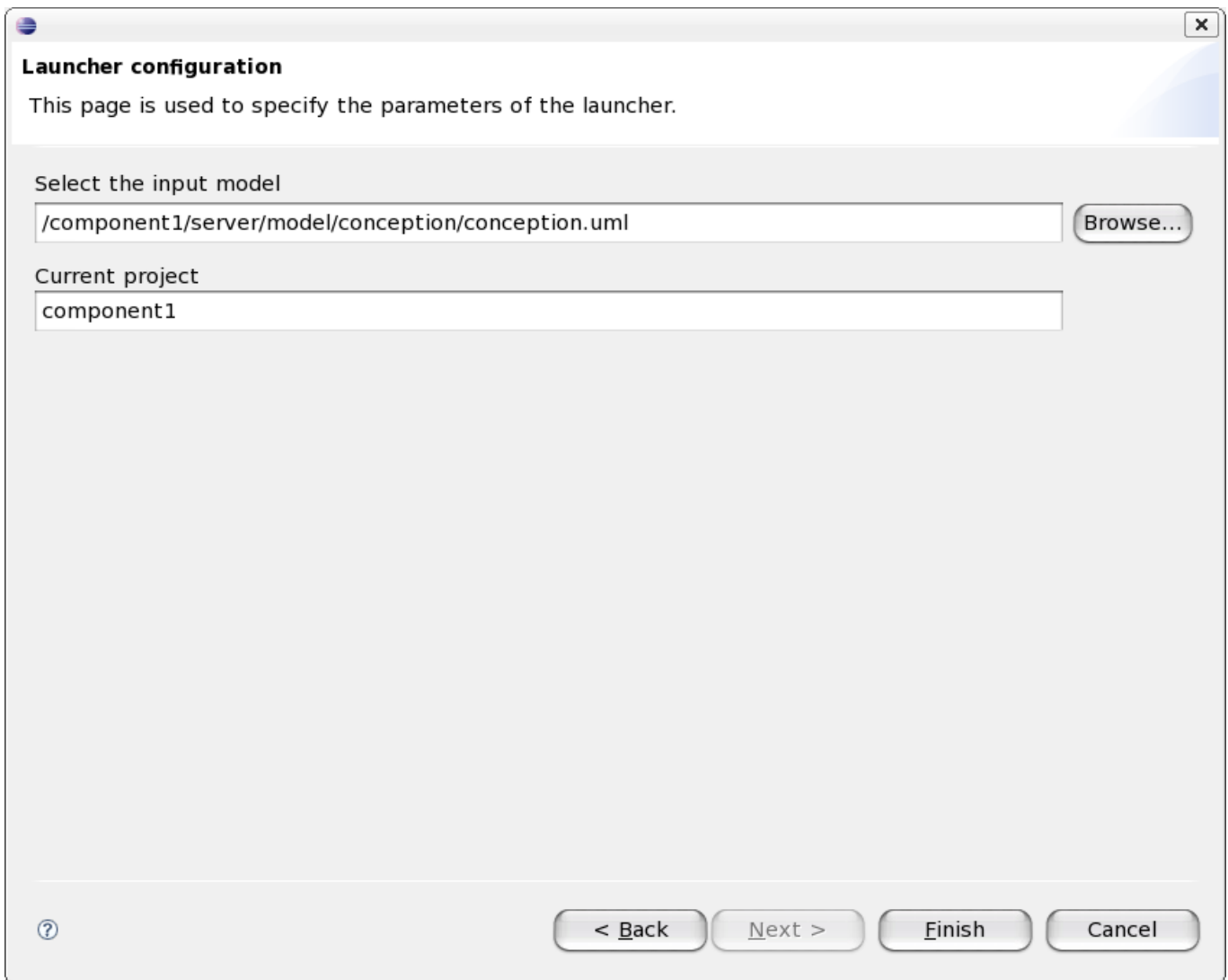


Figure 4.18. Capture de l'écran de l'assistant ModuleLauncher, étape 4

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

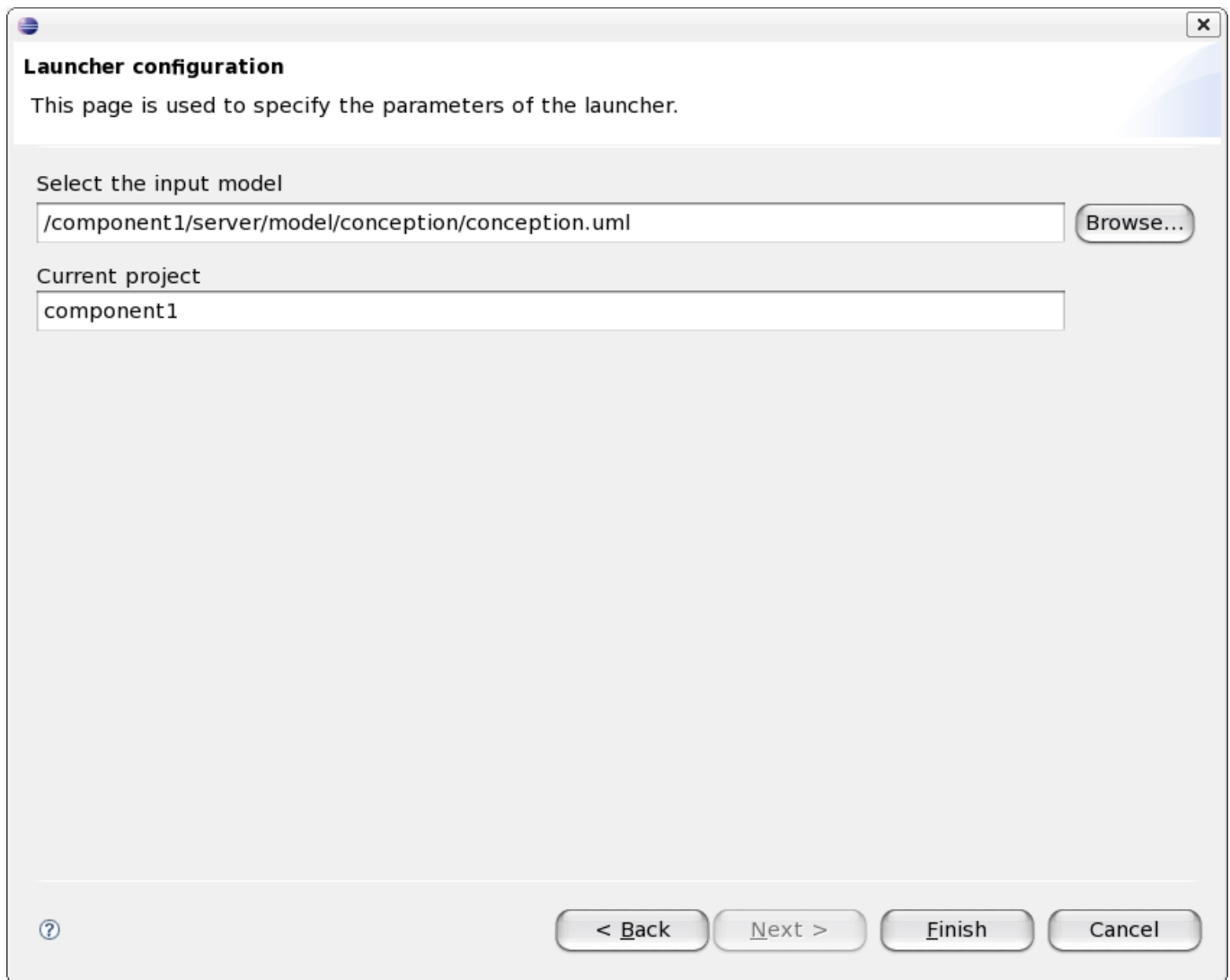


Figure 4.19. Capture de l'écran de l'assistant ModuleLauncher, étape 5

Editer la chaîne en double-cliquant sur le fichier créer.(étape optionnel) Elle contient la référence au modèle ainsi que celle correspond à l'architecture choisie :

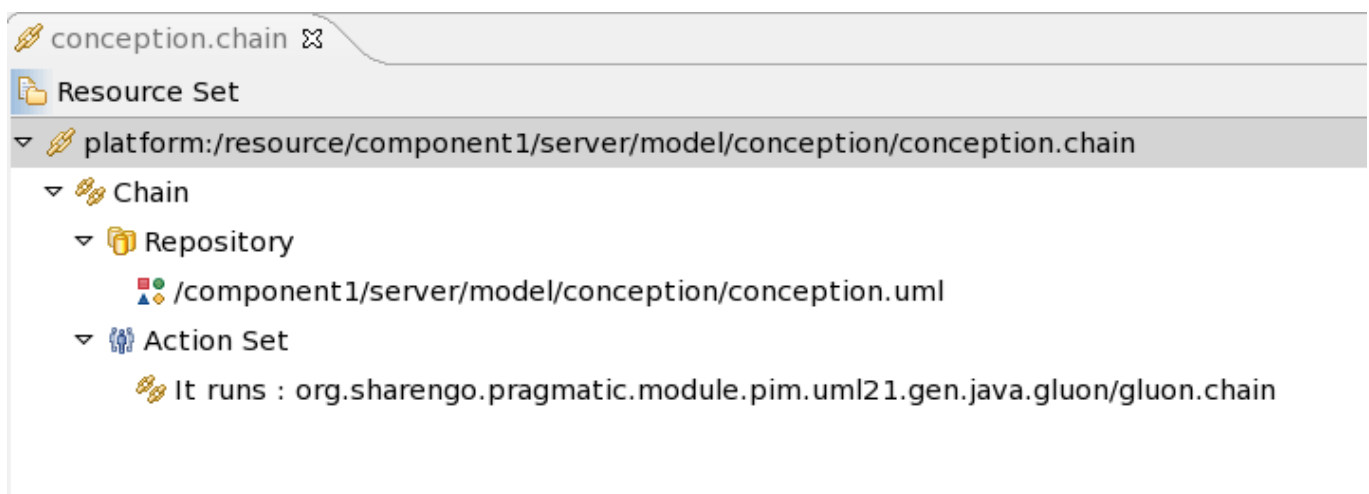


Figure 4.20. Capture de l'écran du contenu de la chaîne

Lancer la génération du code source à l'aide du menu "Launch" dans le menu contextuel du fichier `conception.chain`.

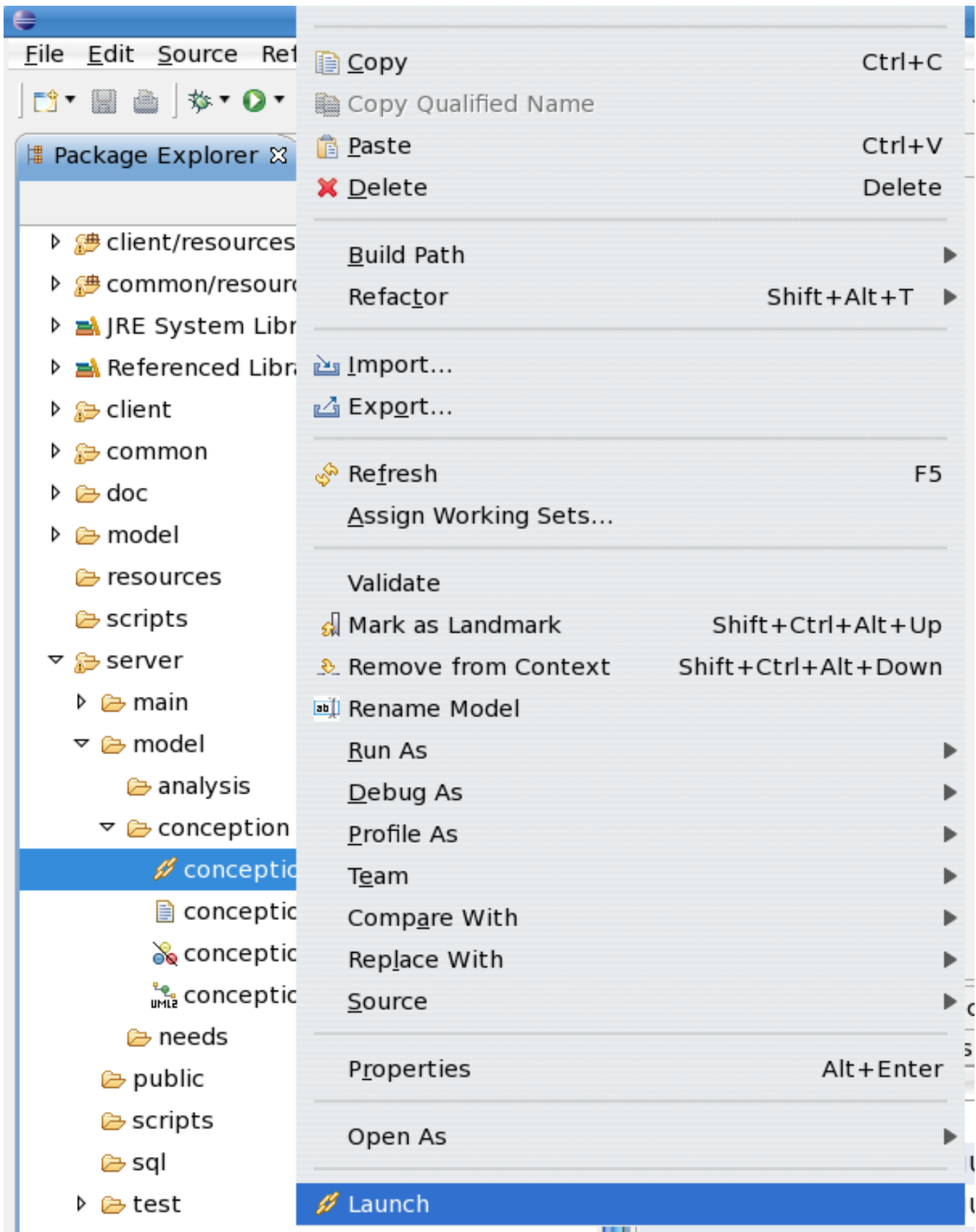


Figure 4.21. Capture de l'écran du lancement d'une chaîne de génération

4.4.1. L'architecture physique basée sur Spring

Spring est une librairie java permettant de construire des socles techniques implémentant différentes architectures. Il s'agit d'un "conteneur léger", une infrastructure similaire à celles offertes par les serveurs d'application JEE, fonctionnant de manière autonome. La grande force de Spring est son un système de fabrique de graphe d'objets basé sur le patron de conception "Inversion Of Control" appelé "IoC" et décrit à cette adresse <http://www.martinfowler.com/articles/injection.html> par le célèbre architecte : Martin Fowler. La documentation de référence du projet est accessible à cette adresse <http://www.springframework.org/docs/reference/>

4.4.1.1. Les pré-requis

Pour utiliser Spring au sein d'un composant ou module, il est impératif d'ajouter une dépendance source vers le composant `middleware/gluon-core`. Ce dernier constitue notre socle technique définissant l'ensemble des dépendances binaires nécessaires au bon fonctionnement de Spring.

Tout d'abord, vous devez récupérer ce composant depuis le référentiel à l'aide du menu `New / Project ... / ABS / Create a new Component from SCM repository` :

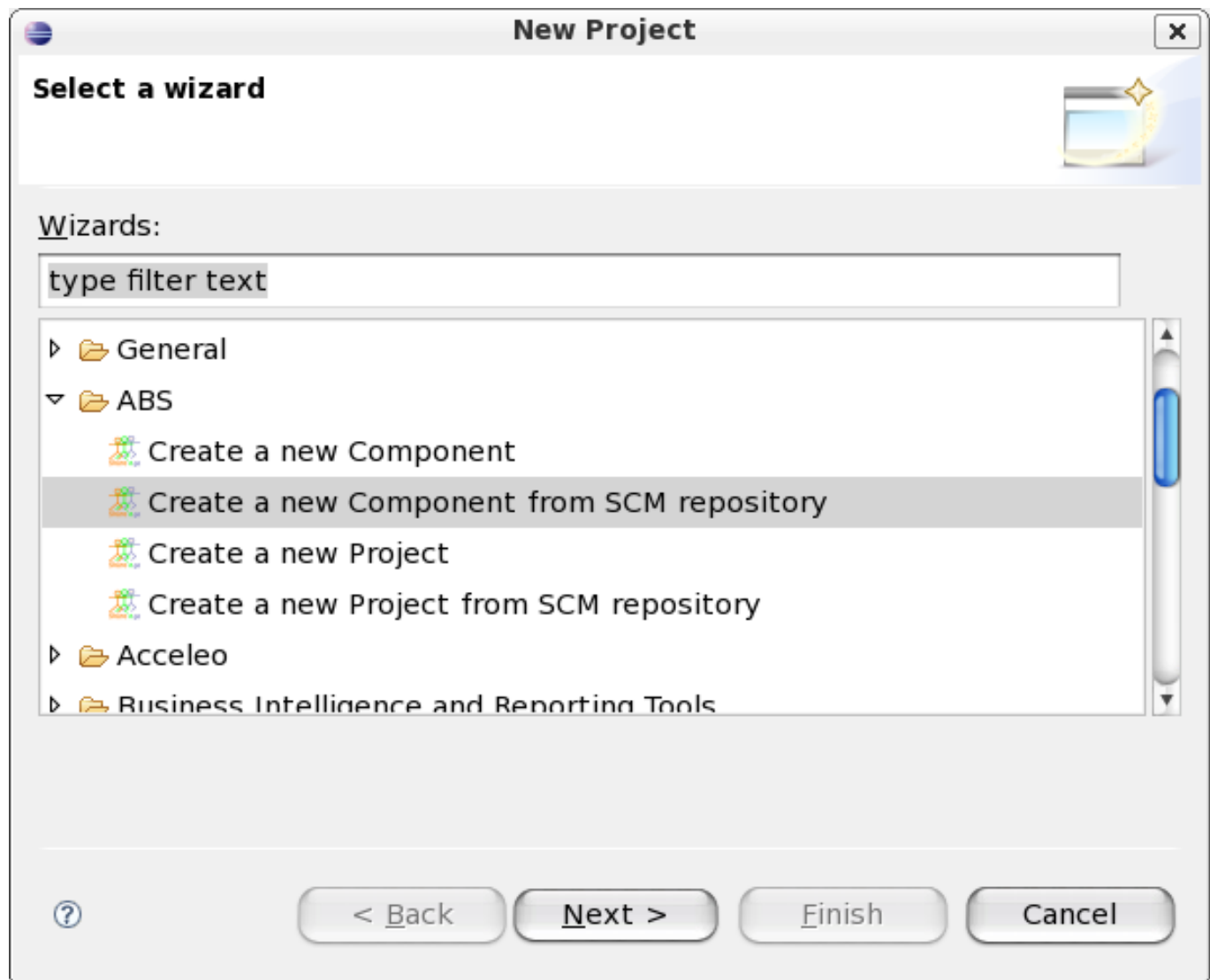


Figure 4.22. Capture de l'écran de récupération d'un composant existant, étape 1

Choisir le référentiel et saisir le nom du composant. Dans le cas présent nous utilisons la version trunk du composant :

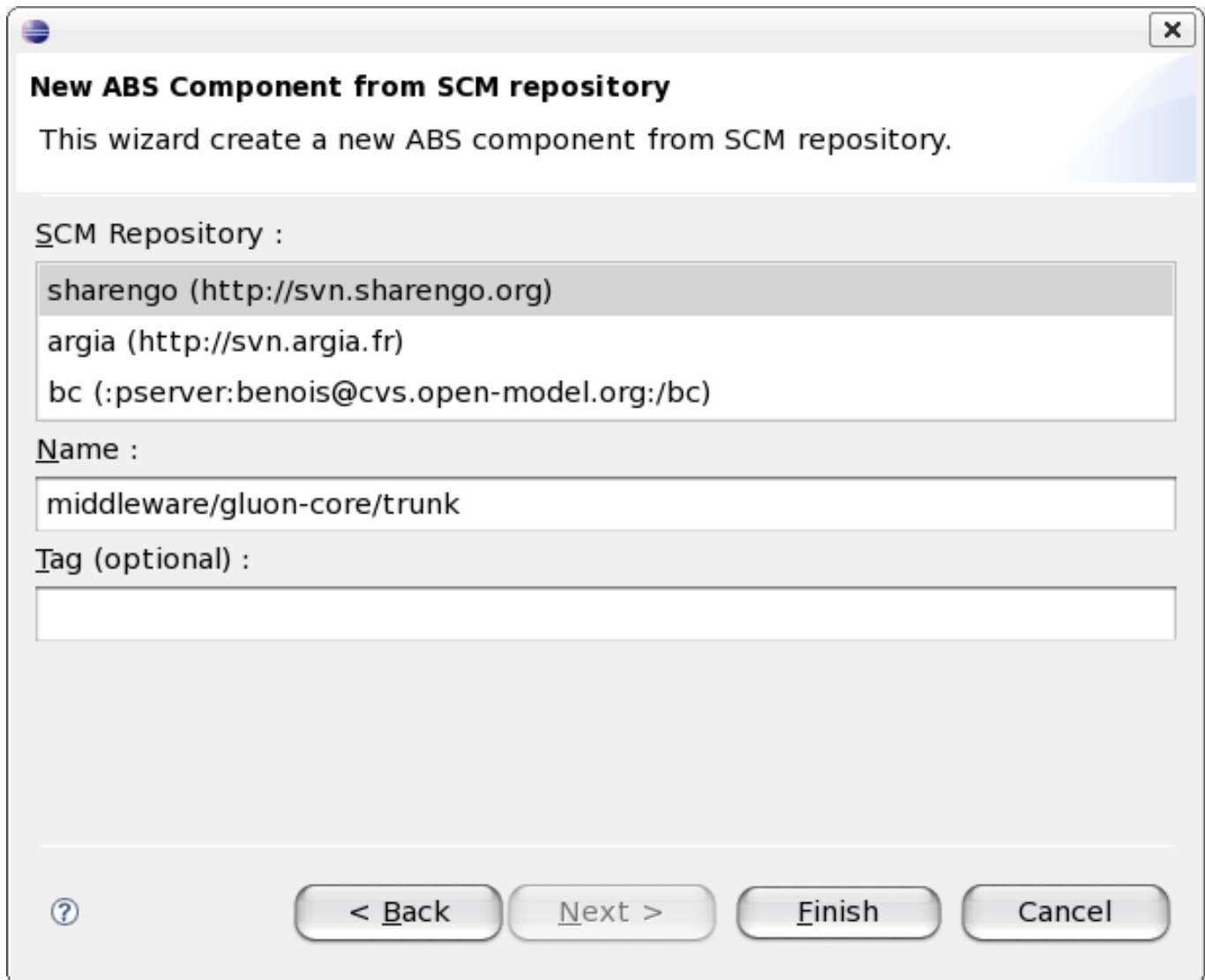


Figure 4.23. Capture de l'écran de récupération d'un composant existant, étape 2

Ensuite ajouter la dépendance vers gluon-core en editant le fichier `srcdep.txt` à la racine de votre composant ou module, et ajouter la ligne suivante :

Exemple 4.1. Un exemple de fichier `srcdep.txt`

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

Note

Si vous souhaitez utiliser des WebServices, vous devez reproduire la même opération afin d'utiliser le composant `middleware/gluon-xfire`.

4.4.1.2. Les fichiers générés

Vous trouverez ci-dessous la liste des artefacts générés pour chacun des stéréotypes :

4.4.1.2.1. A partir du stéréotype <<Entity>>

Pour une classe Entity nommée "MyEntity1" stockée dans le paquetage entities d'un modèle nommé org::sharengo::component1, les fichiers suivants sont générés :

- server/main/src/org/sharengo/component1/entities/MyEntity1.java : classe métier
- server/resources/org/sharengo/component1/entities/hibernate/MyEntity.hbm.xml : fichier de mapping hibernate
- server/test/src/org/sharengo/component1/entities/impl/MyEntityHelper.java : classe utilitaire permettant de créer des instances de cet objet métier. (utilisé dans les tests unitaires)

4.4.1.2.2. A partir du stéréotype <<Dto>>

Pour une classe Dto nommée "MyDto1" stockée dans le paquetage dto d'un modèle nommé org::sharengo::component1, les fichiers suivants sont générés :

- /server/main/src/org/sharengo/component1/dto/MyDto1.java : la classe de transfert

4.4.1.2.3. A partir du stéréotype <<EntitiesManager>>

Pour une classe EntitiesManager nommée "MyEntity1Mgr" stockée dans le paquetage entities d'un modèle nommé org::sharengo::component1, les fichiers suivants sont générés :

- server/main/src/org/sharengo/component1/entities/IMyEntity1Mgr.java : interface d'accès aux données
- server/main/src/org/sharengo/component1/entities/impl/MyEntity1MgrImpl.java : classe d'implémentation des accès aux données
- server/test/src/org/sharengo/component1/entities/impl/MyEntity1MgrTest.java : test unitaire prouvant le bon fonctionnement de l'implémentation
- server/resources/META-INF/spring/org.sharengo.component1/layer-daos-hibernate.xml : fichier de configuration Spring permettant de déclarer l'implémentation courante et ses dépendances requises.

4.4.1.2.4. A partir du stéréotype <<Process>>

Pour une classe Process nommée "MyProcess1" stockée dans le paquetage process d'un modèle nommé org::sharengo::component1, les fichiers suivants sont générés :

- server/main/src/org/sharengo/component1/process/IMyProcess1.java : interface du service métier
- server/main/src/org/sharengo/component1/process/impl/MyProcess1Impl.java : classe d'implémentation du service métier
- server/test/src/org/sharengo/component1/process/impl/MyProcess1Test.java : test unitaire prouvant le bon fonctionnement de l'implémentation

- `server/resources/META-INF/spring/org.sharengo.component1/layer-services.xml` : fichier de configuration Spring permettant de déclarer l'implémentation courante et ses dépendances requises. Il définit également la stratégie transactionnelle associée à chaque opération si le stéréotype Transactional est utilisé.

Si certaines méthodes utilisent le stéréotype Remote, les fichiers suivant sont générés :

- `server/main/src/org/sharengo/component1/process/IMyProcess1WebService.java` : interface contenant uniquement la déclaration des opérations Remote
- `server/resources/META-INF/spring/org.sharengo.component1/layer-xfire-services.xml` : fichier de configuration Spring permettant de lier l'interface à l'implémentation du process et de définir le mapping des URL permettant d'accéder aux opérations
- `server/test/src/org/sharengo/component1/process/impl/MyProcess1WebServiceTest` : test unitaire prouvant le bon fonctionnement des méthodes lors d'un appel distant. Ce test utilise le test unitaire du process en fournissant une instance de process obtenu à l'aide d'un appel distant sur un serveur d'application Jetty embarqué.
- `server/test/src/org/sharengo/component1/process/impl/MockMyProcess1WebService.java` : objet bouchon utilisé par les tests unitaires
- `client/main/src/org/sharengo/component1/ws/WSCClientFactory.java` : fabrique d'objets permettant de récupérer des instance d'`IMyProcess1WebService` à partir d'une adresse du type `http://localhost/project1/services/IMyProcess1WebService`

4.4.1.2.5. A partir du stéréotype <<Controller>>

Pour une classe Controller nommée "MyController1" stockée dans le paquetage ui d'un modèle nommé `org::sharengo::component1`, les fichiers suivants sont générés :

- `server/main/src/org/sharengo/component1/ui/MyController1.java` : classe d'aiguillage vers l'Ui
- `server/resources/META-INF/spring/org.sharengo.component1/layer-controllers.xml` : fichier de configuration Spring permettant de déclarer l'implémentation courante et ses dépendances requises.

4.4.1.2.6. A partir du stéréotype <<Ui>>

Pour une classe Ui nommée "MyUi1" stockée dans le paquetage ui d'un modèle nommé `org::sharengo::component1`, les fichiers suivants sont générés :

- `server/main/src/org/sharengo/component1/ui/IMyUi1.java` : interface de l'Ui
- `server/main/src/org/sharengo/component1/ui/impl/MyUi1Impl.java` : classe d'implémentation de l'Ui
- `server/resources/META-INF/spring/org.sharengo.component1/layer-uis.xml` : fichier de configuration Spring permettant de déclarer l'implémentation courante et ses dépendances requises.

4.4.1.2.7. A partir du stéréotype <<View>>

Pour une classe View nommée "MyView1" stockée dans le paquetage ui d'un modèle nommé

org::sharengo::component1, les fichiers suivants sont générés :

- server/main/src/org/sharengo/component1/ui/impl/MyView1.java : classe vue
- server/main/src/org/sharengo/component1/ui/ViewFactory.java : fabrique d'objet vue, la vue étant un objet à état elle ne peut pas être injecté comme les autres composantes de l'architecture

4.4.1.2.8. A partir du stéréotype <<Config>>

Pour une classe Config nommée "MyConfig1" stockée dans le paquetage config d'un modèle nommé org::sharengo::component1, les fichiers suivants sont générés :

- server/main/src/org/sharengo/component1/Conf/MyConfig1.java : classe de configuration
- server/resources/META-INF/spring/org.sharengo.component1/layer-configs.xml : fichier de configuration Spring permettant de déclarer la classe.

4.4.1.2.9. A partir du modèle

- server/resources/META-INF/spring/component.xml : fichier de configuration Spring permettant de définir quels fichiers layer-* sont requis pour ce composant.
- server/resources/META-INF/spring/component-test.xml : rôle identique à component.xml mais utilisé dans le contexte des tests unitaires et donc paramétrable.
- server/resources/META-INF/spring/org.sharengo.component1/applicationContext-tests.xml : fichier de configuration Spring permettant de définir les ressources hibernate et autres pour les tests unitaires.
- server/test/src/org/sharengo/component1/AbstractBusinessLayerTests.java : classe abstraite utilisée par tous les tests unitaires. Elle permet d'initialiser le graphe de composants.

4.5. Tester le code généré

4.5.1. Configurer les couches techniques

Avant de lancer les tests unitaires, il est nécessaire de vérifier et éventuellement d'ajuster le paramétrage du composant situé dans l'arborescence `server/resources/META-INF/spring` :

4.5.1.1. Le fichier component.xml

Ce fichier est le point d'entrée d'un composant ou module, il permet de définir les pré-requis nécessaires au bon fonctionnement du composant. Il indique les couches techniques et applicatives qui doivent être chargées au démarrage du composant. Vous devez modifier ce fichier en fonction du contexte d'utilisation : composant sans ui, avec ui, facade WebServices, ...

Exemple 4.2. Un exemple de fichier `component.xml`

```

<?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/sc

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

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

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

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

</beans>

```

4.5.1.2. Le fichier `component-test.xml`

Ce fichier charge tous les composants dépendants ainsi que les ressources nécessaires à l'exécution des tests unitaires. A priori le contenu généré dans ce fichier convient dans la plupart des cas, toutefois vous pouvez le modifier.

Exemple 4.3. Un exemple de fichier `component-test.xml`

```

<?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/sc

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

<!-- inclut l'ensemble des fichiers component.xml présent dans le classpath pour demarrer tous les compos
<import resource="classpath*:META-INF/spring/component.xml"/>

<!-- charge les ressources nécessaire a l'execution des 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. Le fichier applicationContext-tests.xml

Il se trouve dans le répertoire `server/resources/META-INF/spring/NOM_DU_MODELE/` et spécifie les ressources base de données et autres si besoin. Il peut-être nécessaire de compléter la section "Start of user code other hbm files" afin de charger les fichiers de mapping présent dans un autre composant.

Exemple 4.4. Un exemple de fichier applicationContext-tests.xml

```
<?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. Tester la couche d'accès aux données

La classe `server/test/src/org/sharengo/component1/entities/impl/MyEntityMgrTest.java` implémente un test unitaire permettant de prouver les opérations basiques d'accès aux données c'est-à-dire les opération des sauvegardes, mis à jour et restauration des données. Elle doit être complété pour toutes opérations ajoutée sur l'EntitiesManager dans la modèle. Lancer le test unitaire :

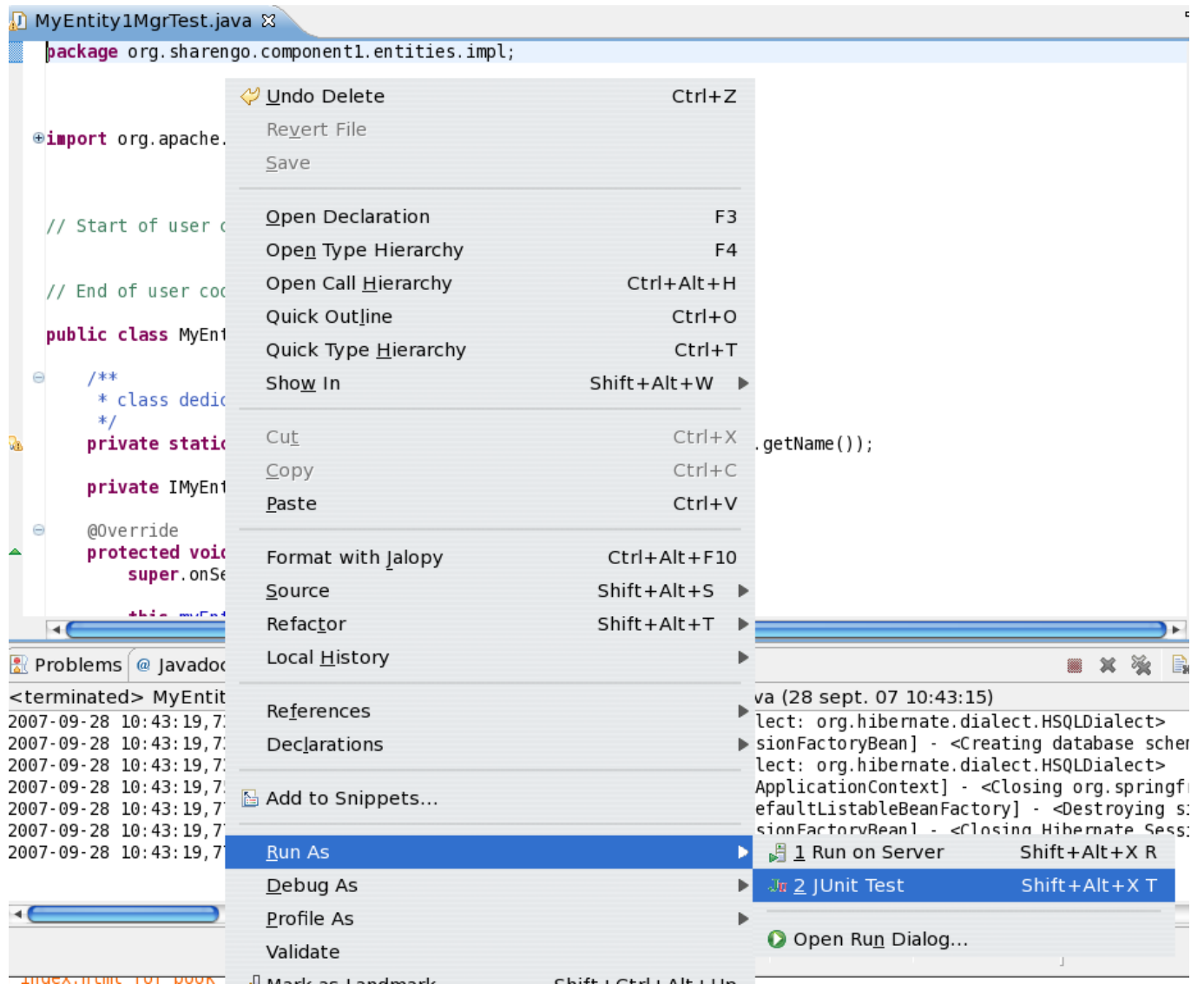


Figure 4.24. Capture de l'écran du lancement du test unitaire

Consulter le résultat :

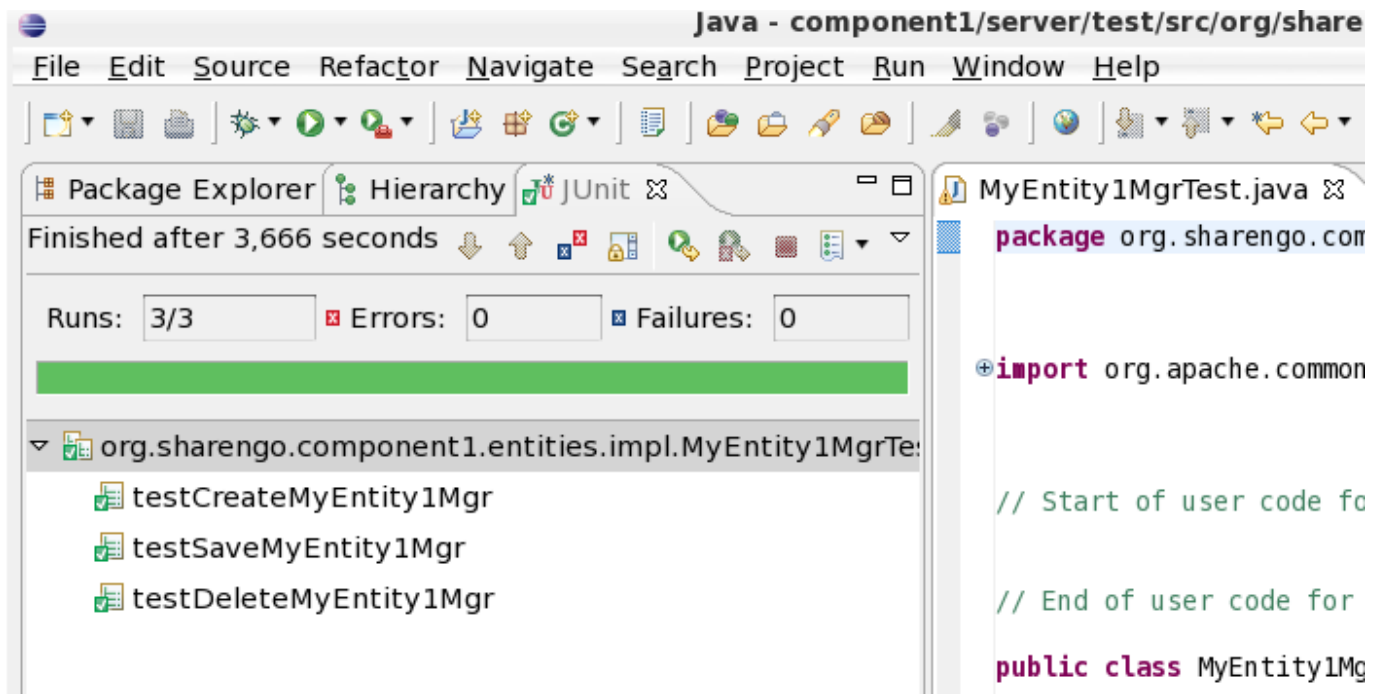


Figure 4.25. Capture de l'écran du résultat du test unitaire

4.5.3. Tester les process métiers

La classe `server/test/src/org/sharengo/component1/process/impl/MyProcess1Test.java` implémente un test unitaire permettant de prouver le bon fonctionnement du process métier. Coder le test, le process et lancer le test de la même manière.

4.5.4. Tester les Web Services

La classe `server/test/src/org/sharengo/component1/process/impl/MyProcess1WebServiceTest.java` implémente un test unitaire permettant de prouver le bon fonctionnement du process métier en mode WebService. Lancer le test de la même manière afin de valider les éventuels problème liés à ce mode de communication.

4.6. Lancer le projet dans tomcat

4.6.1. Configurer votre projet pour le déployer en tant qu'application Web

La variable `${PROJECT_NAME}` doit être remplacé par le nom de votre projet dans les fichiers d'exemples.

4.6.1.1. Le fichier web.xml

Le fichier web.xml est généré automatiquement par l'assemblage des différents descripteurs issus des composants. Cette opération est réalisée par la tâche de déploiement "deploy:build.war". Le plugin se charge d'appeler cette tâche à chaque démarrage de tomcat, vous n'avez donc rien à configurer.

4.6.1.2. Configuration de l'accès à la base de donnée

Vous devez définir la clé suivante `@@hibernate.datasource@@` dans `${PROJECT_NAME}/deployments/localhost/server/replace.server.properties`

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

Pensez à ajouter le driver JDBC dans le répertoire `tomcat/common/lib` et editez le fichier `${PROJECT_NAME}/modules/main/server/main/conf/context.xml` afin de définir les paramètres d'accès à la base de données :

```
<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>
```

Créez et éditez le fichier suivant `${PROJECT_NAME}/modules/main/server/main/conf/applicationContext.xml` afin de configurer hibernate en déclarant la liste des fichiers de mapping à charger :

```
<?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>
```

Créez la base PostgreSQL associée :

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

Ajouter les dépendances nécessaires vers les composants utilisés par le projet dans le fichier `srcdep.txt` du module du projet.

4.6.2. Lancer Tomcat depuis Eclipse

Démarrer le serveur d'application Tomcat à l'aide du menu contextuel "Run As / Run On Server" du déploiement :

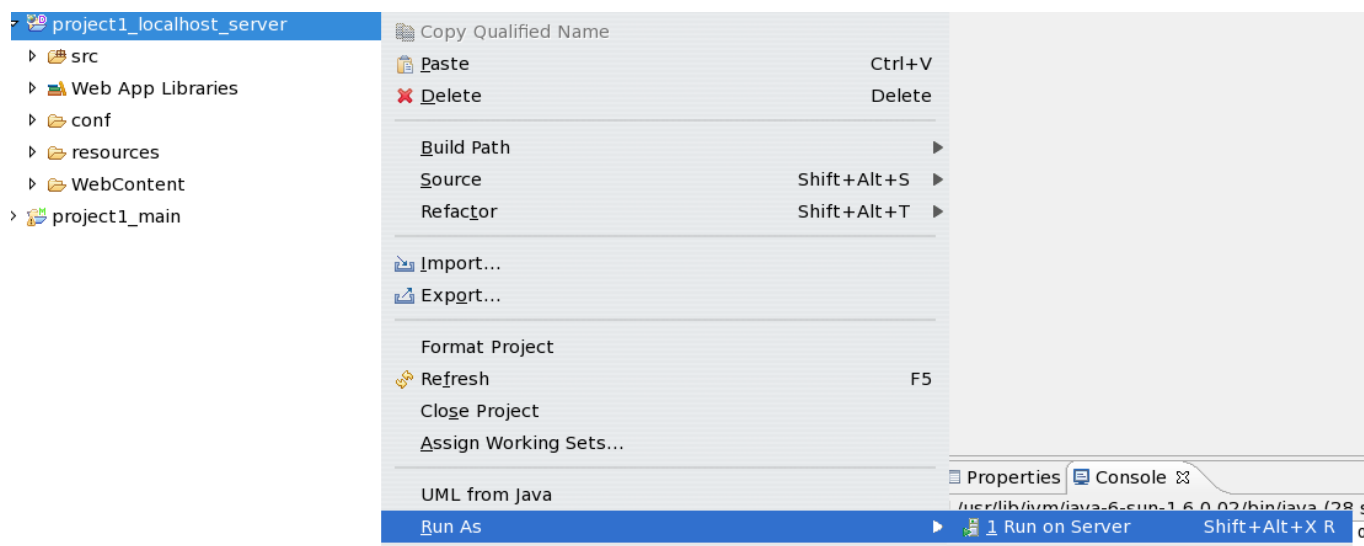


Figure 4.26. Capture de l'écran du démarrage de tomcat sous eclipse, étape 1

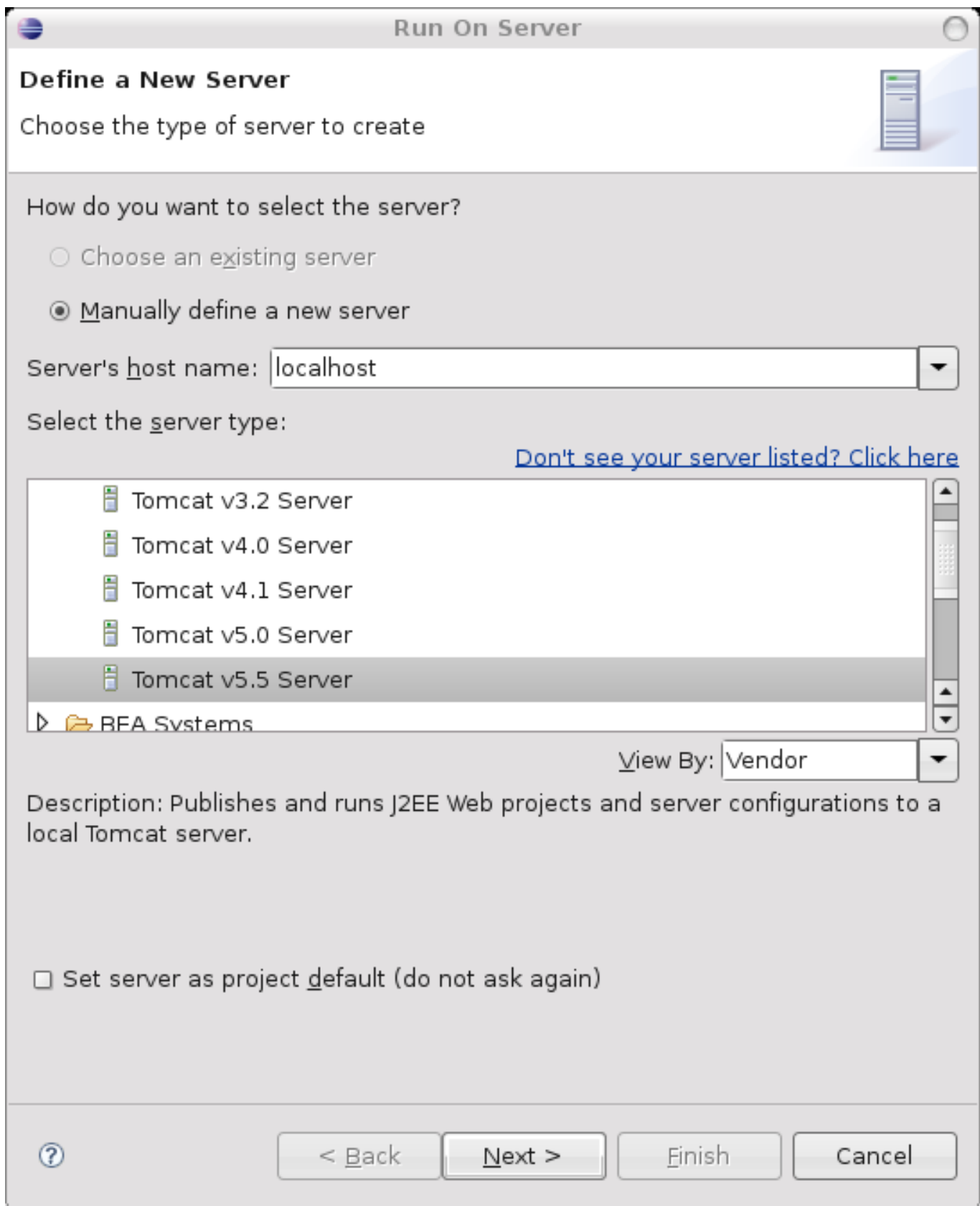


Figure 4.27. Capture de l'écran du démarrage de tomcat sous eclipse, étape 2

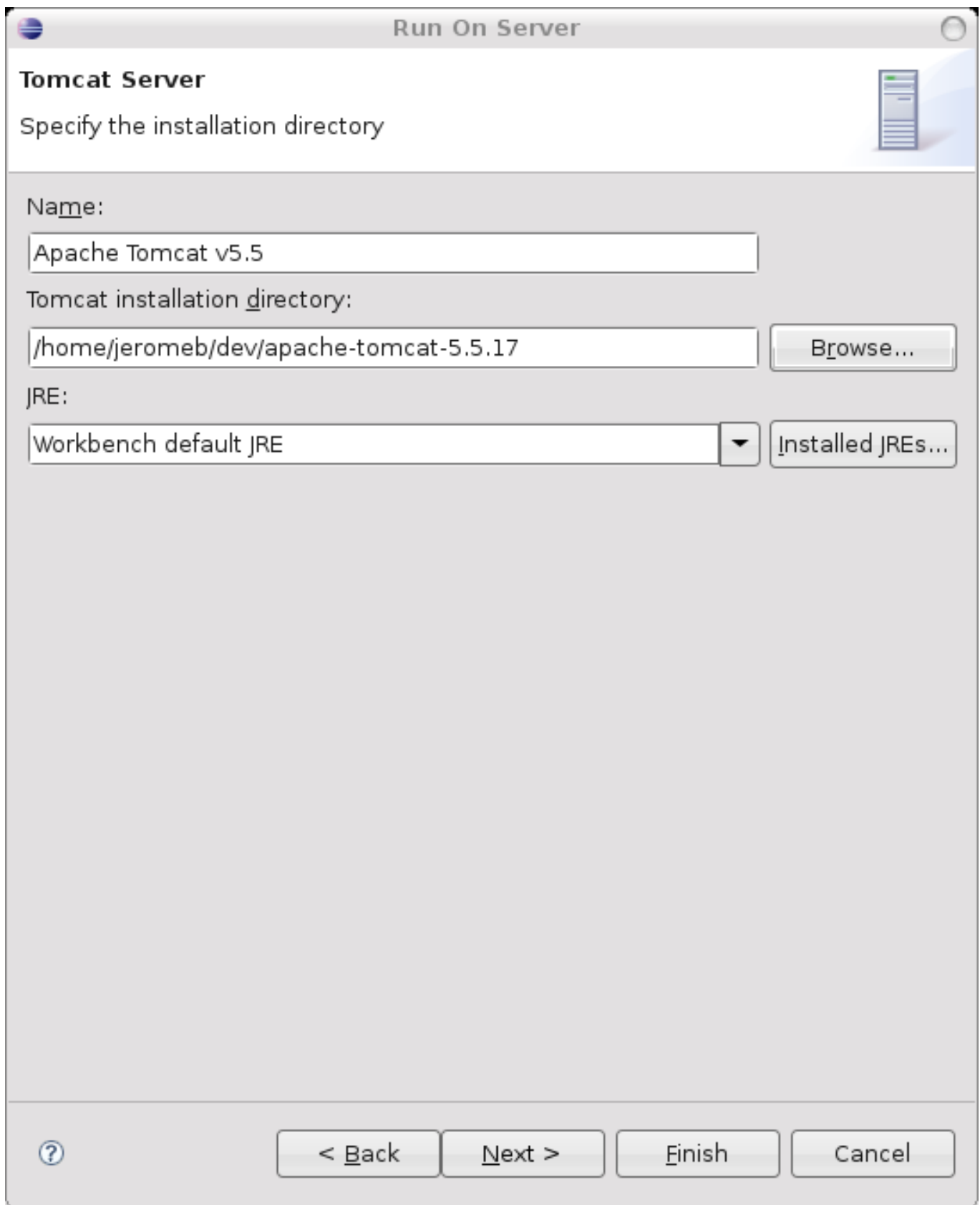


Figure 4.28. Capture de l'écran du démarrage de tomcat sous eclipse, étape 3